Dynamic Load Balancing Strategies in Heterogeneous Distributed System

by

Bibhudatta Sahoo

Department of Computer Science & Engineering
National Institute of Technology, Rourkela
Rourkela-769008, Odisha, INDIA

2013

Dynamic Load Balancing Strategies in Heterogeneous Distributed System

Thesis submitted in partial fulfilment
of the requirements for the degree of

Doctor of Philosophy *in*

Computer Science & Engineering

by

Bibhudatta Sahoo
(Roll No: 50406001)

Under the guidance of

Prof. Sanjay Kumar Jena
&
Prof. Sudipta Mahapatra

Department of Computer Science & Engineering
National Institute of Technology, Rourkela
Rourkela-769008, Odisha, INDIA

2013

National Institute of Technology,Rourkela,India

Certificate

This is to certify that the work in the thesis entitled **"Dynamic Load Balancing Strategies in Heterogeneous Distributed System"** submitted by **Bibhudatta Sahoo** is a record of an original research work carried out by him under our supervision and guidance in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy in Computer Science and Engineering during the session 2013-2014 in the department of Computer Science and Engineering, National Institute of Technology Rourkela. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Date:
Place: NIT Rourkela

Dr. Sanjay Kumar Jena
Professor
Department of Computer Science & Engineering
National Institute of Technology
Rourkela-769008, India

Date:
Place:IIT Kharagpur

Dr. Sudipta Mahapatra
Associate Professor
Department of Electronics &
Electrical Communication Engineering
Indian Institute of Technology
Kharagpur-721302, India

Acknowledgements

I would like to express my sincere gratitude to my supervisors *Dr. Sanjay Kumar Jena* and *Dr. Sudipta Mahapatra*, for their enthusiastic and effective supervision that made this work possible. Deep gratitude to my doctoral scrutiny committee members *Dr. S. K. Rath, Dr. K. K. Mahapatra* and *Dr. B. Majhi* for their valuable suggestions and supportive counsel to shape this thesis work. I sincere regard to all my teachers who taught me in the department of CSE NIT Rourkela.

I would like to express my thanks to my faculty colleagues of the CSE department for their peer support. It is a pleasure to acknowledge the help and support of *Prof. Ashok kumar Turuk* and *Prof. K. Satya Babu*. I am specially thank full to the research scholars in particular, *Dilip kumar, Sukanta, Supreet, Sumit*, and *Sovagya* for their assistance and help. I must offer a blanket thank you to all the faculty and staff with whom I have worked at NIT Rourkela. I am thankful to Prof. S. S. Mohapatra, Prof. S. K. Patra, Prof. U C. Pati, Prof. R. BaliarSingh for their encouragement and moral support during the thesis work.

The thesis is based on the published research results from various researchers in the field of distributed system. Sincerer attempt have been made to present the materials in our own words and have given credit to the esteemed researchers and the original source of information. Special thanks to my friends *Rajkisore, Punyaban* and *Umesh* for their valuable support in various ways to support this thesis work.

Last but not least, I am forever indebted to my parents and in-laws for their inspiration and blessings when it was most needed. Finally, I wish to express my sincere appreciation to my wife *Nibedita* for her constant encouragement and my little daughter *Stutee*, who is my source of inspiration. Must thank my family, have provided me with the love, stability, and practicality that has allowed me to persist with studying. Such a connection is valuable in itself. This thesis is dedicated to them.

Bibhudatta Sahoo
NIT Rourkela

Abstract

Distributed heterogeneous computing is being widely applied to a variety of large size computational problems. This computational environments are consists of multiple heterogeneous computing modules, these modules interact with each other to solve the problem. Dynamic load balancing in distributed computing system is desirable because it is an important key to establish dependability in a Heterogeneous Distributed Computing Systems (HDCS). Load balancing problem is an optimization problem with exponential solution space. The complexity of dynamic load balancing increases with the size of a HDCS and becomes difficult to solve effectively. The solution to this intractable problem is discussed under different algorithm paradigm.

The load submitted to the a HDCS is assumed to be in the form of tasks. Dynamic allocation of n independent tasks to m computing nodes in heterogeneous distributed computing system can be possible through *centralized* or *decentralized* control. In *centralized approach*,we have formulated load balancing problem considering task and machine heterogeneity as a linear programming problem to minimize the time by which all task completes the execution in *makespan*.

The load balancing problem in HDCS aims to maintain a balanced allocation of tasks while using the computational resources. The system state changes with time on arrival of tasks from the users. Therefore, heterogeneous distributed system is modeled as an $M/M/m$ queue. The task model is represented either as a consistent or an inconsistent expected time to compute (ETC) matrix. A batch mode heuristic has been used to design dynamic load balancing algorithms for heterogeneous distributed computing systems with four different type of machine heterogeneity. A number of experiments have been conducted to study the performance of load balancing algorithms with three different arrival rate for the task. A better performance of the algorithms is observed with increasing of heterogeneity in the HDCS.

A new codification scheme suitable to *simulated annealing* and *genetic algorithm* has been introduced to design dynamic load balancing algorithms for HDCS. These stochastic iterative load balancing algorithms uses sliding window techniques to select a batch of tasks, and allocate them to the computing nodes in the HDCS. The proposed dynamic genetic algorithm based load balancer has been found to be effective, especially in the case of a large number of tasks.

Approximation algorithms have been used to design polynomial time algorithms for intractable problems that provide solutions within the bounded proximity of the optimal solution. Analysis and design of two approximation algorithms based on task and machine heterogeneity has been presented with *makespan* as performance metric. The two proposed approximation schemes have been compared with an optimal solution computed as *lower bound* and are proved to be *2-approximation* and *3/2 approximation* algorithm.

The decentralized load balancing problem in heterogeneous distributed systems is modeled as a multi-player non-cooperative game with Nash equilibrium. In the process prior to execute a task, the heterogeneous computing nodes are participate in a non-cooperative game to reach an equilibrium. Two different types of decentralized load balancing problems are presented in this thesis as minimization problems with *price*, *response time*, and *fairness index* as the performance metric. These algorithms are used to design decentralized load balancing strategies to minimize the cost of the entire computing system.

We have proposed eight new centralized load balancing algorithms that operates in batch mode and two decentralized load balancing algorithm. These algorithms are applicable to our proposed linear programming problem formulation for load balancing in a HDCS. The centralized algorithms have been presented in three groups as per algorithmic paradigm. As distributed systems continue to grow in scale, in heterogeneity, and in diverse networking technology, they are presenting challenges that need to be addressed to meet the increasing demands of better performance and services for various distributed application. All theses proposed load balancing algorithms are tested with nodes and task availability and found to be effective with different performance metric.

Contents

List of Figures

List of Tables

List of Symbols

Symbols	Description
n	Number of task to be executed
m	Number of computing nodes available in the system
$M = \{M_1, M_2, ..., M_m\}$	Set of m computing nodes in the system
P_j	Processor with computing node M_j
π_j	Main memory with computing node M_j
S_j	Secondary memory with computing node M_j
μ_j	Service rate or processing rate of node M_j
λ	Arrival rate of the task to the system
$T = \{t_1, t_2, ..., t_n\}$	Set of n tasks in the system
t_{ij}	Expected time to compute task t_i on node M_j
at_i	Arrival time of task t_i to the system
C_{ij}	Expected earliest completion time of the task t_i on node M_j
ϕ	Empty set
AT	Set of arrival time for n tasks
HAT	Mean heap of task as per arrival time
A(j)	Set of task assigned to node M_j
L_j	The load on node M_j
CL_j	Current load of node M_j
L	Makespan
L_{min}	Lower bound of load balancing problem
M/M/1	Single server queue with exponential arrival and service rate
M/M/m	m server queue with exponential arrival and service rate

Symbols	Description
TS	Task Schedule:
p_i	Probability that i number of tasks in the system
a_i	Probability that the t_i is allocated to node M_j
λ_j	Arrival rate of the task to node M_j
$\rho_j = \frac{\lambda_j}{\mu_j}$	Traffic intensity or average utilization of node M_j
s_j	Local scheduler with computing node M_j
Q_j	Task queue associated with computing node M_j
p_{ji}	Price per unit of resources available in M_j for computing task t_i
r_{ij}	Fraction of task migrated from Q_i to node M_j
R_j	Load balancing strategy for scheduler s_j on M_j
$R = \{R_1, R_2, ..., R_m\}$	load allocation vector for heterogeneous distributed computing system
$T_j(R)$	Expected response time at computing node M_j
$D_j(R)$	Total expected response time at node M_j
$\psi_j(R)$	Expected cost of computing at node M_j
D	Total expected response time of the system
I(D)	Fairness index of the system

List of Abbreviations/Acronyms

Abbreviations/Acronyms	Description
AU	Average utilization
CPU	Central processing unit
ETC	Expected time to compute matrix
ECT	Expected computing time
FCFS	First-come, first-served
FF	First-fit
GA	Genetic algorithm
GAs	Genetic algorithms
GOSP	Global optimal scheme with pricing
GPU	Graphics processing unit
HDCS	Heterogeneous distributed computing system
HPC	High performance computing
I/O	Input/output
LPP	Linear programming problem
NASHP	Nash scheme with pricing
NP	Non-deterministic polynomial
SETC	Sorted expected time to compute matrix
SA	Simulated annealing
TS	Task schedule
TIG	Task Interaction Graph

Chapter 1

Introduction

Parallel and Distributed computing over the past three decade witnessed phenomenal growth due to the declined cost of hardware, advancement in communication technology, explosive growth of internet and need to solve large-scale problems. The resources in distributed computing systems should be allocated to the computational tasks, so as to optimize some system performance parameter that guarantee a user specified level of system performance. In particular, the load balancing is concerned with resource allocation policies to assign tasks to computing nodes. The load balancing of distributed computing system becomes a major research issue to utilize the ideal computing resources. The thesis addresses different aspects of dynamic load balancing issues in heterogeneous distributed computing system with task and machine heterogeneity

1.1 Introduction

Distributed computing systems are built over a large number of autonomous computer nodes. These computing nodes are uniquely identified in a network with their IP address and interconnected by SANs, LANs, or WANs in a hirarchical manner [1]. Distributed computing platforms are designed to deliver parallel computing environment for various potential computing and non-computational problems. The potential of distributed computing systems are related to the management and allocation of computing resources relative to the computational load of the system [2, 3, 4, 5, 6]. There has been a phenomenal growth in the number of Internet users and variety of applications in the Internet. Heterogeneous Distributed Computing System (HDCS) utilizes a distributed suite of different high-performance machines, interconnected with high-speed links, to perform different

computationally intensive applications that have diverse computational requirements. Modern distributed computing technology includes clusters, the grid, service-oriented architectures, massively parallel processors, pear-to-pear networking, and cloud computing [1]. Distributed computing provides the capability for the utilization of remote computing resources and allows for increased levels of flexibility, reliability, and modularity.

Heterogeneous computing is the coordinated use of different types of machines, networks and interfaces to maximize their combined performance [7, 8, 9]. In a heterogeneous distributed computing system the computational power of the computing entities are possibly different for each node. The advanced architectural feature of the node can be exploited by the task to meet the computational requirement. The HDCS structure provides information on computing system and communication network to the users. The applicability and strength of HDCSs are derived from their ability to match the computing need of a task, and allocate them to appropriate resources. However, the large computing power remains unexploited to a greater extent because of the lack of software systems and tools for managing the resources. Scheduling problems mainly addresses the allocation of distributed computing resources over time to the tasks that are parts of the process running in the system.

There are numerous applications that run on top of a distributed system. The services provided by these applications are grouped into two category, *data intensive* and *computation intensive*. Some of the popular applications that are using distributed systems are: world wide web(WWW), network file server, banking network, peer-to-peer networks, process control systems, sensor networks, grid computing, and cloud services. In general, HDCS environments are well suited to meet the computational demands of large, diverse groups of tasks. Hence, the area of research is of interest among the researchers from industry and academia.

1.2 Distributed computing system

A distributed system [1, 10, 11, 12, 13] is a collection of multiple autonomous computers each having its own private memory, interconnected through a computer network, and capable of collaborating on a task. The functional capabilities of a distributed system are based on multiple processes, interprocess communication, disjoint address space and a collective goal. In computer architecture terminology, these implementation belongs to the class of loosely coupled MIMD machines, with each node having a private address space. Distributed systems can also be implemented on a tightly couple MIMD machine, where processes running on separate processors are connected to a globally shared mem-

ory [12].

A large heterogeneous distributed computing system consists of potentially millions of heterogeneous computing nodes connected by the Internet. The applicability and strength of HDCS are derived from their ability to meet computing needs by the use of appropriate resources [5, 14, 15]. Heterogeneity in distributed computing system can be expressed by considering three systems attributes (i) processor with computing node, (ii) memory, and (iii) networking [14]. Performance metric used to quantify the processing power of the processor or node by means of processing speed and represented with Floating point Operations per Second (FLOPS). LINPACK is being used as the benchmark to quantify the processing capability of a node, and expressed in FLOPS [16]. The computing nodes in this thesis is a computer, a workstation, a cluster, or a supercomputer that can be identified by an unique address to the rest of the world. The computing power of a node is expressed in FLOPS. Memory attributes are measured as the available memory capacity to support the process. The networking attributes are the link capacity associated with transmission medium, propagation delay and available communication resources [17]. Heterogeneity of architecture and configuration complicates the load balancing problem [5].

A distributed application consists of a set of task with certain relation among them. Tasks are the basic units handled by an HDCS. We have assumed an application in an HDCS to be either a task or represented by a Task Interaction Graph(TIG). The execution time of an application may be influenced by a number of parameters, which are either application dependent or system dependent [18]. The major factors considered for task execution are, *(i) communication topology induced by the application*, *(ii) the work load assigned to the computing nodes*, *(iii) computing capability of the node* and *(iv) inter node communication cost*. Heterogeneity can also arise due to the difference in *task arrival rate* at homogeneous processors or processors having different task processing rates. Large degrees of heterogeneity in an HDCS adds significant additional complexity to the scheduling problem [19]. It is possible to minimize the total execution time in a HDCS by considering the node to which a task can be assigned and the cost of communication that results from task assignment.

Each individual computing node has a local scheduler, If the scheduling decisions are left to the individual nodes without global coordination it results in distributed scheduling. Using distributed schedulers, each node can take an independent decision to execute the task or the task is to be migrated to another node. This decentralized mechanism are more suitable for dynamic load-balancing of a large-scale distributed computing en-

vironment than centralized mechanisms in term of scalability and fault tolerance [20, 21]. the central scheduling model is based on interleaving of actions and coordination with local schedulers of every computing node in the HDCS. The central scheduler, also known as a serial scheduler or load balancing service, is able to effectively control the computing resources for dynamic allocation of the tasks in a distributed system [20]. A single computing node that acts as a central scheduler or resource manager of the distributed computing system collects the global load information of other computing nodes. The resource management sub systems of the HDCS are designated to schedule the execution of the tasks dynamically as that arrives for the service. However, a central scheduler exhibits poor parallelism and poor scalability. In practice each of the schedulers finds its own importance in distributed resource management and is used by the researchers in representation and design of distributed algorithms.

There are number of techniques and methodologies for scheduling processes of a distributed system. These are task assignment, load balancing, and load-sharing approaches [22, 23]. In the *task assignment approach*, each process submitted by a user for processing is viewed as a collection of related tasks and these tasks are scheduled to suitable nodes so as to improve performance. A *load sharing approach* simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed. In a *load balancing approach*, tasks submitted by the users are distributed among the nodes of the system so as to equalize the workload among the nodes at any point of time. Task might have to be migrated from one machine to another even in the middle of execution to ensure equal workload. Load balancing strategies may be static or dynamic [6, 17, 22].

1.3 Load balancing in distributed system

Load balancing is a crucial issue in parallel and distributed systems to ensure fast processing and optimum utilization of computing resources. The load of a computing node is measured as sum of the expected time to compute (ETC) of the individual tasks [24, 25]. Load imbalance in a distributed computing system is due to the fluctuations in arrival and service patterns. Due to this a task waits for execution in a node while other nodes are ideal [26]. The *load imbalance factor* quantifies the degree of load imbalance within a distributed computing system. A decision on load balancing is made when *load imbalance factor* is grater than *load balancing overhead* at a particular time. Load balancing strategies try to ensure that every processor in the system does almost the same amount of work at any point of time. There are a number of techniques and methodologies for scheduling

processes of a distributed system. These are task assignment, load-balancing, and load-sharing approaches [5]. Load balancing is a common approach to task assignment in a distributed system such as web server farms, database systems, grid computing clusters, and others [27]. A load balancing algorithm has to make use of the system resources in such a manner that resource usage, response time, network congestion, and scheduling overhead are optimized. Due to heterogeneity of computing nodes, jobs encounter different execution times on different processors. In the task assignment approach, each process submitted by a user for processing is viewed as a collection of related tasks and these tasks are scheduled to suitable nodes so as to improve performance. A load sharing approach simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed. In a load balancing approach, processes submitted by the users are distributed among the nodes of the system so as to equalize the workload among the nodes at any point of time. Processes might have to be migrated from one machine to another even in the middle of execution to ensure equal workload.

Load balancing strategies may be static or dynamic [5, 6, 28, 29, 30]. *Static* strategies are based on advance information governing the load balancing decision. The *dynamic* load balancing strategies allocate the tasks to the computing nodes based on their current state. Dynamic load distribution (also called load balancing, load sharing, or load migration) can be applied to restore balance [29]. In general, dynamic load-balancing algorithms can be broadly categorized as *centralized* or *decentralized* [15, 31] according to how these are implemented. These are divided into reactive and predictive load balancing strategies [15, 31]. Centralized algorithms use the central or serial scheduler to schedules the tasks in a distributed system using the load information available from other computing nodes. Decentralized algorithms are implemented with control mechanisms distributed to each computing node of the distributed computing system. The allocation decisions are the result of exchange of load information between the computing nodes. To improve the utilization of the computing node, parallel computations require that tasks be distributed to the nodes in such a way that the computational load is spread evenly among the nodes.

A large amount of supporting research has been reported in the area of static and dynamic load balancing on distributed computing system. Due to the potentially arbitrary nature of the load arrival and departure process, dynamic load balancing is substantially more challenging than static load balancing [32]. Dynamic load balancing on HDCS research covers a wide range of system models across homogeneous and heterogeneous architectures suitable to applications. These applications range across embedded real-

time systems, commercial transaction systems, transportation systems, and military or space systems - to name a few. The supporting research includes system architecture, design techniques, performance metric, queuing model, simulation framework, iterative load balancing algorithms, theory, testing, validation, proof of correctness, modeling, software reliability, operating systems, parallel processing, and real-time processing. The performance of an HDCS can be improved by proper task allocation and an effective scheduling policy.

The load balancing problem is an optimization problem with an exponential solution space. The solution space is defined as the collection of all possible solutions for a given problem. The optimization algorithms are the search algorithms, that are used to find the optimal solutions from the search space. The load distribution problem is known to be NP-hard [33]. Moreover, the complexity of dynamic load balancing increases with the size of the HDCS and becomes difficult to solve effectively [34]. The load balancing problem has been evenly treated, in both the fields of computer science and operations research.

Dynamic load balancing algorithms are characterized by six policies: initiation, transfer, selection, profitability, location and information [5, 6, 35, 36].

i. *Initiation policy:* decides who should invoke the load balancing activity.

ii. *Transfer policy:* determines if a node is in a suitable state to participate in load transfer.

iii. *Selection policy:* source node selects most suitable task for migration.

iv. *Profitability policy:* a decision on load balancing is made based on load imbalance factor of the system at that instant.

v. *Location policy:* decides which nodes are most suitable to share the load.

vi. *Information policy:* provides a mechanism to support load state information exchange between computing nodes.

An extensive methodology has been developed in this field over the past thirty years. A number of load balancing algorithms have been developed, dealing with homogeneous and heterogeneous distributed system on different work load models. The design of load balancing algorithms, in general considers the underlying network topology, communication network bandwidth, and task arrival rate at each of the node in the system [37]. Wu [5] has suggested nine different quantifiable design parameters for load balancing algorithms. Those parameters of a distributed system are listed below:

i. *System size:* number of nodes in the system.

ii. *System load:* load on each node.

iii. *System traffic intensity:* arrival rate of task to computing node.

iv. *Migration threshold:* load level to initiate task migration.

v. *Task size:* size of the task suitable for migration effort.

vi. *Overhead cost:* costs for task migration.

vii. *Response time:* turnaround time for a task.

viii. *Load balancing horizon:* number of neighbouring nodes to be probed to finalize the task destination.

ix. *Resource demand:* demands on system resource by a task.

To summarize, load balancing in HDCS can be defined by combining system architecture and the particular application with certain quality of service. Moreover, both centralized and decentralized dynamic load distribution are desirable because of the applications running on various modern distributed computing system like clusters, the grid, service-oriented architecture, massively parallel processors, and peer-to-peer system, and cloud.

1.4 Literature review

Load balancing for distributed computing system is a problem that has been deeply studied for a long time. Casavant and Kuhl [28] have characterized the structure and behavior of decision-making policies, in particular referring to the load sharing policies considering performance and efficiency. Xu and Lau [38] presented a classification of iterative dynamic load balancing strategies in multicomputer concern with task migration from a computing node to nodes across nearest neighbour. Shivaratri *et al.* [39] provides a survey and taxonomy of load sharing algorithms based on the design paradigm. Boyer *et al.* [40] presented load balancing dealing with heterogeneity and performance variability. Analysis of load balancing strategy for the web server cluster system has been presented in [41].

Kremien and Kramer [42] have presented an quantitative analysis for distributed system that provides both performance and efficiency measures considering load and the

delay characteristic of the environment. The task allocation problem of the distributed computing system has been presented as a *0-1 quadratic programming problem* by Yin *et al.* [43]. They have designed a hybrid particle swarm optimization algorithm for finding the near optimal task allocation within a reasonable time. A mixed integer linear programming formulation for minimizing set up cost for multipurpose machine is presented in [44]. A linear programming formulation representing task allocation model for maximizing reliability of a distributed system can be found in [45]. Altman *et al.* [46] investigated optimal load balancing policy for multi-class multi-server systems and a Poisson input stream and with heterogeneous service rates for centralized and distributed decentralized non-cooperative systems. The probability of load imbalance in heterogeneous distributed computer system have been studied by Keqin Li [47] with a method to minimize the probability of load imbalance in the system.

Queueing model can be viewed as key models for the performance analysis and optimization of parallel and distributed systems [48]. Modeling of optimal load balancing strategies using queuing theory was proposed by Francois Spies [29]. This is one of the pioneer works reported in the literature that presents an analytical model of dynamic load balancing techniques as an $M/M/k$ queue and simulates it with fundamental parameters like load, number of nodes, transfer speed and overload rate [29]. Queuing-theoretic models for parallel and distributed system can also be found in [48, 49]. The most appropriate queuing model for homogeneous distributed system is an $M/M/m/n$ queue [50]. General job scheduling problem of n tasks with m machines is presented as an optimization problem in [49] to minimize the makespan. Makespan measures the maximum time by which all n tasks complete their execution in m machines. Nicola *et al.* [51] have developed $M/G/1$ queuing models to derive the distribution of job completion time in a failure-prone environment where the system changes with time according to events such as failures, degradation, and/or repair. An adaptive load sharing techniques for queue control in order to achieve optimal or near optimal efficiency and performance has been discussed by Kabalan *et al.* [52].

A variety of distributed system model have been used by the researchers to present the dynamic load balancing problem. Techniques for mapping tasks to machines in HDCS, considering task and machine heterogeneity is reported in [53] for static and dynamic heuristics. In dynamic resource allocation scenarios the responsibility for making global scheduling decisions are lie with one centralized schedular, or are shared by multiple distributed schedulers [54]. Hence, dynamic load balancing algorithms can be further classified into a centralized approaches and a decentralized approaches. In a centralized approach [6, 55, 56] one node in the distributed system acts as the central controller and

is responsible for task allocation to other computing nodes. The central controller takes the decision based on load information obtained from the other nodes in the distributed system.

In a decentralized approach [39] all the computing nodes participated in the task allocation process. This decentralized decision making can be realized through a cooperation or without cooperation among the computing nodes. The related research work on dynamic load balancing presented in the following sections are grouped into *(i) centralized load balancing* and *(ii) decentralized load balancing.*

1.4.1 Centralized dynamic load balancing

Gopal *et al.* [3] presented a simulation study for four load balancing algorithm on heterogeneous distributed systems with a central job dispatcher. A different form of linear programming formulation of the load balancing problem has been discussed along with greedy, randomized and approximation algorithms to produce sub-optimal solutions to the load balancing problem. A mono-population and hybrid genetic based scheduling algorithm has been proposed by Kolodziej and Khan [57] to schedule independent jobs to minimize *makespan* and flow time. A minimized *makespan* central scheduler considering the cost of communication has been presented by Tseng *et al.* [58] in the dynamic grid computing environment. Bekakos *et al.* [59] discussed the generic resource sharing in a grid computing platform. Li *et al.* [60] presented a centralized load balancing scheme for sequential tasks on grid environment to achieve minimum execution time, maximum node utilization and load balancing among the nodes. A new load metric called *number of effective tasks* has been developed by Choi *et al.* [61] to design a dynamic load balancing algorithm for the workstation clusters. A randomized dynamic load balancing algorithm framework along with convergence proof are being discussed in [62]. The impact of heterogeneity on scheduling independent tasks on a centralized platform is analyzed in [63] with the objective of minimizing the *makespan*, maximize the response time and the sum of all response times. A performance evaluation approach to compare different distributed load balancing schemes can be found in [64]. A comparative analysis of centralized scheduling policies combining processor and I/O scheduling has been presented by Karatza [65]. Scheduling of applications represented as Task Iteraction Graphs (TIGs) and Directed Acyclic Graphs (DAGs) on heterogeneous computing systems can be found in [45, 66, 67, 68]. A centralized task assignment policy suitable for a multiple-server firm is presented by Jayasinghe *et al.* [69] based upon pre-emptive migration of tasks. Solomon *et al.* [70] have presented a collaborative multi-swarm particle swarm optimization for task matching in heterogeneous distributed computing environments. A general

model is presented in [71] for a centralized heterogeneous distributed system to study distributed service reliability and availability. Terzopoulos and Karatza [72] evaluated centralized load balancing algorithms by varying the arrival rate of task in heterogeneous clusters. Dynamic load balancing algorithm based on task classification has been presented by Wang *et al.* [73]. A particle swarm optimization based Load-Rebalance Algorithm for Task-Matching in Large Scale Heterogeneous Computing Systems has been addressed by Sidhu *et al.* [74].

1.4.2 Decentralized dynamic load balancing

A dynamic decentralized load balancing algorithm for computationaly intensive jobs on HDCS has been proposed by Lu *et al.* [75]. A truthful mechanism for solving the static load balancing problem in heterogeneous distributed system was addressed by Grosu and Chronopoulos [76]. A predictive decentralized load balancing approach complemented through CORBA can be found in [77]. Decentralized load distribution policies without preemption, in non-dedicated heterogeneous clusters and grids, are presented using three different queueing discipline [78]. Economides and Silvester [79] have formulated and solved the load sharing, routing and congestion control problem in arbitrary distributed systems using game theory. A decentralize task scheduling strategy for multiple classes of tasks in a heterogeneous system has been presented by Qin and Xie [80] with a new metric to quantify system availability and heterogeneity. A decentralized load distribution scheme has been invented by Lakshmanan *et al.* [81]. Chakraborty *et al.* [82] have used congestion game theoretic models to address the load balancing problem in a distributed environment.

Number of researchers have used multi-agent based system for resource allocation in distributed computing environment. A multi- agent task allocation model can be found in [83]. However, both the centralized and distributed dynamic load balancing algorithms are equally important considering the problems and applications that requires a distributed computing system.

1.5 Motivation and research challenges

Distributed computing systems have become increasingly popular as cost effective alternative to traditional high performance computing platform [84]. The main aim of load balancing problem on heterogeneous distributed computational environments is an efficient mapping of tasks to the set of computing nodes. The dynamic load balancing problem remains a challenging global optimization problem due to the: (*i*) *heterogeneous*

structure of the system, (*ii*) *computing resources administrative domains* and (*iii*) *Quality of Service(QoS) requisitions by applications.* As suggested in lecture notes on *approximation algorithm* by Motwani [85] it is usually hard to tell the exact difference between an optimal solution and a near-optimal solution. Hence, it seems reasonable to devise algorithms which are really efficient in solving *NP*-hard problems, at the cost of providing feasible solutions which in all cases is guaranteed to be only sub-optimal. The dynamic load balancing problem considering heterogeneity of the computing system has been reported rarely in the related research work. The major motivation that leads to study dynamic load balancing strategies in HDCS are listed as:

- The computing capability of HDCS can be exploited by designing efficient task allocation algorithms that assign each task to the best suitable computing node for execution.
- Due to heterogeneity of computing nodes, jobs encounter different execution times on different computing nodes. Therefore, research should address scheduling in heterogeneous environments.
- As distributed systems continue to grow in scale, in heterogeneity, and in diverse networking technology, they are presenting challenges that need to be addressed to meet the increasing demands of better performance and services for various distributed application.
- Because of the intractable nature of the task assignment problem on HDCS, it is desirable to obtain a best-possible solution through the design of new strategies for dynamic load balancing in HDCS.
- The tasks and computing resources could be dynamically added and dropped to and from the system. This necessitates dynamic load balancing algorithms that use system-state information for load assignment.

The problem of finding an assignment of task to the computing nodes that results in minimum *makespan* is $NP-$hard [33]. The most common approach used by the researchers to find solutions to $NP-$hard problems are treating them with *integer programming tools,* or *heuristics,* or *approximation algorithm* [86, 87]. Scheduling in HDCS remains a challenging global optimization problem due to the heterogeneous structure of the system, co-existence of locally and geographically dispersed job dispatchers and resource owners that usually work in different autonomous administrative domains. Hence, some of the research challenges in devising dynamic resource allocation policies for heterogeneous environments are listed as:

- To design system model for heterogeneous distributed computing system
- To model dynamic load balancing problem as optimization problems that can be appropriate to a variety of HDCS.
- To design queuing models, which can used as the key model for performance analysis and scheduling in distributed system considering machine heterogeneity.
- To design dynamic resource allocation strategy in heterogeneous computing environment considering task and machine heterogeneity for the different task arrival rate.
- Heuristic algorithms may produce good solutions against the quality of the solution, whereas approximation algorithm have the capability to produce solution, which are guaranteed to be within a bound. So, it is a challenge to design approximation polynomial time algorithms for an intractable load balancing problem that provide solutions within the bounded proximity of the optimal solution.
- HDCS are highly scalable and regularly increasing with user base with the ownership from distinct individuals as organizations requires a *decentralized resource management system*. So research challenge is to design decentralize load balancing strategies with the involvement of all of the computing nodes in an HDCS.

1.6 Problem statement

The problem addressed in this thesis are the research challenges highlighted in the previous section. The load submitted to the HDCS is assumed to be in the form of tasks. Dynamic allocation of n independent tasks to m computing nodes in heterogeneous distributed computing system is possible through centralized or decentralized control. The load balancing problem in HDCS aims to maintain a balanced execution of tasks while using the computational resources.

The load balancing problem using a *centralized approach* which is formulated considering task and machine heterogeneity, is presented in Section 2.6 as a linear programming problem to minimize the time by which all complete execution or (*makespan*) in a HDCS.

The decentralized load balancing problem in heterogeneous distributed systems is modelled as a multi player non-cooperative game with Nash equilibrium. Prior to executing a task, the heterogeneous computing nodes participate in a non-cooperative game to reach an equilibrium. Two different types of decentralized load balancing problems are presented in Section 6.4 as minimization problems with either of *price*, *response time*, or,

fairness index as performance metric. One is to minimize the total expected response time of the system and the other is to minimize the total price agreed between the scheduler and computing nodes to execute the set of task.

Since no optimal load balancing strategy exists for dynamic load balancing problem, we resolve to design strategies to obtain sub-optimal solutions using different algorithmic paradigms.

1.7 Research contribution

The generalized load balancing problem can be viewed as assignment of each of n independent jobs on m heterogeneous distributed computing nodes. The Load balancing problem has been formulated for a Heterogeneous Distributed Computing System (HDCS) considering the system and task heterogeneity and presented as an optimization problem with the objective to minimize the *makespan*. The dynamic task assignments are carried out by the central scheduler considering the load of each computing nodes at the instant. The thesis uses three different algorithmic approaches namely (i) greedy algorithm,(ii) iterative heuristic algorithm,and (iii) approximation algorithm, to obtain sub-optimal solutions for load balancing problem.

Four greedy resource allocation algorithms using batch mode heuristic has been presented for heterogeneous distributed computing system with four different type of machine heterogeneity. A number of experiments has been conducted to study the performance of these load balancing algorithms with three different arrival rate for the consistent and inconsistent task model.

Two stochastic iterative load balancing algorithms have been designed with sliding window techniques to select a batch of tasks from the task pool, and allocates them to the computing nodes in a HDCS. A new codification scheme suitable to *simulated annealing* and *genetic algorithm* has been introduced to design dynamic load balancing algorithms for a HDCS.

Approximation algorithms have been used to design polynomial time algorithms for intractable problems that provide solutions within the bounded proximity of the optimal solution. Analysis and design of two approximation algorithms based on task and machine heterogeneity has been presented with *makespan* as a performance metric. A non-cooperative decentralized game-theoretic framework has been used to solve the dynamic load balancing problem as an optimization problem. In decentralize approach of load balancing, all computing nodes in HDCS are involved in load balancing decisions. The decisions to allocate the resources in a HDCS are based upon the pricing model of

computing resources using a bargaining game theory. Two different decentralized load balancing problem presented in this thesis as minimization problems with *price*, *response time*, and *fairness index*. Two algorithms have been proposed to compute load fraction. These algorithms are used to design decentralized load balancing strategies to minimize the cost of the entire computing system leading to load balanced.

1.8 Thesis organization

The present thesis is organized into seven chapters. **Chapter 1** presents introduction and importance of load balancing problem on HDCS with task and machine heterogeneity. It also includes a review of related work. **Chapter 2** presents model for Heterogeneous Distributed Computing System, with the system architecture and workload model. Load balancing problem is presented as a minimization problem with the objective to minimize the *makespan*. An algorithmic approach to the load balancing problem with task and node heterogeneity is presented in this chapter. **Chapter 3** presents a queueing model for the HDCS and analyses the impact of heterogeneity with greedy resource allocation algorithms. Four different types of machine heterogeneity are considered for consistent and inconsistent ETC matrix models. In **Chapter 4** a new codification scheme suitable to SA and GA has been introduced to design dynamic load balancing algorithms for the HDCS. The effect of a genetic algorithm based dynamic load balancing scheme has been compared with first-fit, randomized heuristic and simulated annealing algorithms through simulation. **Chapter 5** presents the analysis and design of centralized approximation algorithms based on task and machine heterogeneity through ETC matrix on HDCS with *makespan* as the performance metric. The proposed approximation scheme has been compared with the optimal solution computed as a lower bound. The load balancing problem in heterogeneous distributed systems is modelled as a multi player non-cooperative game with Nash equilibrium and load balancing strategies using the non-cooperative game theory has been presented in **Chapter 6**. Performance of two existing price-based job allocation schemes, namely Global Optimal Scheme with Pricing (GOSP) and Nash Scheme with Pricing (NASHP), have been analyzed and modified versions of theses schemes have been introduced to analyze the performance by considering the effect of pricing on system utilization. **Chapter 7** concludes the work done, highlighting the contributions and suggests the directions for possible future work on load balancing.

Chapter 2

Heterogeneous Distributed System Model and Algorithemic Framework for Resource Allocation

This chapter introduces the basic concepts, the terminology and the state of the art of the dynamic load balancing problem in heterogeneous distributed computing systems. A model has been presented for a Heterogeneous Distributed Computing System (HDCS) including the system architecture and the workload model. The dynamic load balancing problem is presented as a minimization problem with the objective of minimizing the makespan. An introduction to algorithmic approach to load balancing problem is discussed to solve load balancing problem with task and node heterogeneity.

2.1 Introduction

Distributed systems are loosely coupled and do not have a global clock driving all the nodes. Major atomic components of the distributed systems are the processors, communication network, clocks, software, and non-volatile storage or secondary storage. The key properties of distributed systems are the scalability and autonomous nature of various nodes. Heterogeneous computing systems ranges from diverse computing elements or paradigms within a single computer, to a cluster of different type of Personal Computers, to coordinated geographically distributed computing nodes with different architectures [88]. An abstract model of the HDCS has to be created in order to formalize the system behavior of a heterogeneous distributed computing system. The abstract model of HDCS

describes the system architecture and the workload model. Dynamic load balancing policies may be further sub-divided into centralized and distributed structures according to the use of information about the computing nodes for assigning tasks to the nodes. A detail classification is listed in [28] as a hierarchical taxonomy. In distributed computing systems maximizing utilization of resources is a major concern. Hence, the tasks are assigned/mapped and migrated among computing nodes so that the overall performance and utilization of the system can be optimized. The dynamic load balancers in the centralized controlled systems are able to defer the task allocation until the best computing node is ascertained and/or until one of the suitable computing node becomes ready to receive the task for execution [31].

The present thesis suggests system models, which can be used to represent different distributed computing infrastructures such as network of workstations, commodity of workstations, web server clusters, grids, server firms, and high performance computing clusters. The heterogeneity in distributed computations are mostly influenced by the continuous advancement in micro-electronics technology and communication technology using highly efficient computer networks. We have presented two different heterogeneous distributed system models with centralized and distributed control for resource allocation.

Heterogeneity can also arise due to the difference in task arrival rate at homogeneous processors or processors having different task processing rates. There are a large variety of heterogeneous distributed systems, all of which have certain common characteristics that differentiate them from homogeneous distributed system. Here, we briefly look at heterogeneous distributed system components and system models in the context of dynamic load balancing.

2.2 HDCS model and assumptions

Heterogeneous computing systems are the set of diverse computing resources that can be on a chip, within a computer, or on a local or geographically distributed network [89]. The heterogeneity in computing systems are mostly due to the frequent dynamic developments in computer hardware, software, protocols, application programming interface, communication networks, mobile computing devices and operating system. We consider an HDCS consisting of m independent, heterogeneous, and uniquely addressable computing entities (also termed as computing nodes or processors) as shown in Figure 2.1. Let M be the set of m computing nodes, denoted as $M = \{M_1, M_2, ..., M_m\}$. The system consists of m *heterogeneous* nodes, which represent the heterogeneous distributed computing system (HDCS). Each node has three prime resources, processor, main memory,

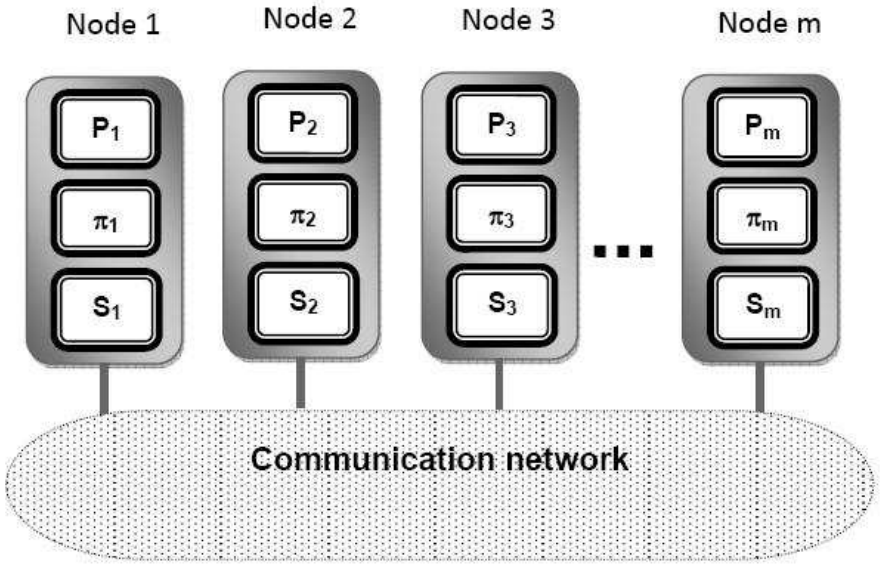

Figure 2.1: Heterogeneous Distributed Computing System

and secondary memory, identified as P_j, π_j and S_j for the node M_j. In the HDCS model all computing nodes are with heterogeneous service capacities. The task execution on node M_j can be characterized as the service rate or processing rate, is denoted as μ_j and exponentially distributed with a mean $\frac{1}{\mu_j}$. The total computing power of the system is $\mu = \Sigma_{j=1}^{m} \mu_j$.

In our study we define an heterogeneous distributed system as follows;

Definition 2.1 (Heterogeneous Distributed System)**.** *An heterogeneous distributed system consists of computing nodes connected by a interconnect, where the messages transfer time between the nodes is bounded. Each node may execute a system kernel that provides the local and remote inter-process communication and synchronization as well as the usual process management functions and input/output management.*

2.3 Computing node model

The HDCS is a scalable computing infrastructure, integrating various hardware, software and network technologies. A computing node in a distributed computing system is an autonomous computer having its own private memory, communicating through a computer

network [1]. A computing node in a distributed system can be a personal computer or workstation or a high performance computing cluster. In general the computing power of a node is dominated by the processor architecture.

A node in a HDCS has three components that are involved in task execution. The *local scheduler* is responsible for scheduling of tasks that arrive at the computing node. The *dispatcher* invokes the next task to be executed on the node following a scheduling policy. The *global scheduler* interacts with the scheduler of other nodes in order to perform load distribution among other nodes in the HDCS. We assume the following regarding the computing nodes:

- Each node M_j is autonomous, has full information on its own resource, and it manages its work load.
- Each node is characterised by its processing rate and only its true value is known to M_j.
- Each node M_j has a processing rate μ_j.
- Each node handles its own communication and computation overheads independently.
- Each node incurs a cost proportional to its utilization.
- Each computing node is always available for processing.
- The computing nodes in a HDCS do not perform multitasking.

Heterogeneity of service is a common feature of many real world multi server queueing situations. These heterogeneous service mechanisms are invaluable scheduling methods that allow the task to receive different quality of service [90]. In practice computing nodes with different architectures and operating systems provides different processing capabilities. Each of the computing nodes is modelled as a single-queue single-server queuing system with an FCFS service discipline. The queue lengths in each node are assumes to be large enough so that the probability of overflow is negligible. The local task arrival to the arbitrary node M_j is assumed to be Poisson with a mean arrival rate λ_j. The computing capability of the node is represented as a service rate of the node, and is assumed to be exponentially distributed with a mean $1/\mu_j$ [55]. Hence, the performance of a node without load sharing can be calculated using the $M/M/1$ queuing model. The $M/M/1$ queueing model represents a single server system with exponential job arrival times and exponential job service times [76, 91].

2.4 System models for dynamic load distribution

The scheduling problem in HDCS aims to maintain a balanced execution of tasks while using the computational resources with computing nodes. Dynamic resource allocation in HDCS is possible through centralized or decentralized control. A dynamic load distribution algorithm must be general, adaptable, stable, scalable, fault-tolerant and transparent [5].

2.4.1 Centralized system model for HDCS

A centralized dynamic load balancing algorithm operates based on the load information from other computing nodes and can be realized through a centrally controlled HDCS. A centralized model of HDCS consists of a set $M = \{M_1, M_2, ..., M_m\}$, of m independent heterogeneous, and uniquely addressable computing nodes as shown in Figure 2.2, with one node acts as the resource manager. The single computing node that acting as a central scheduler or resource manager of the system is responsible for collecting the global load information of other computing nodes. Resource management sub-systems of the HDCS are designated to allocate the tasks to the computing nodes for their execution.

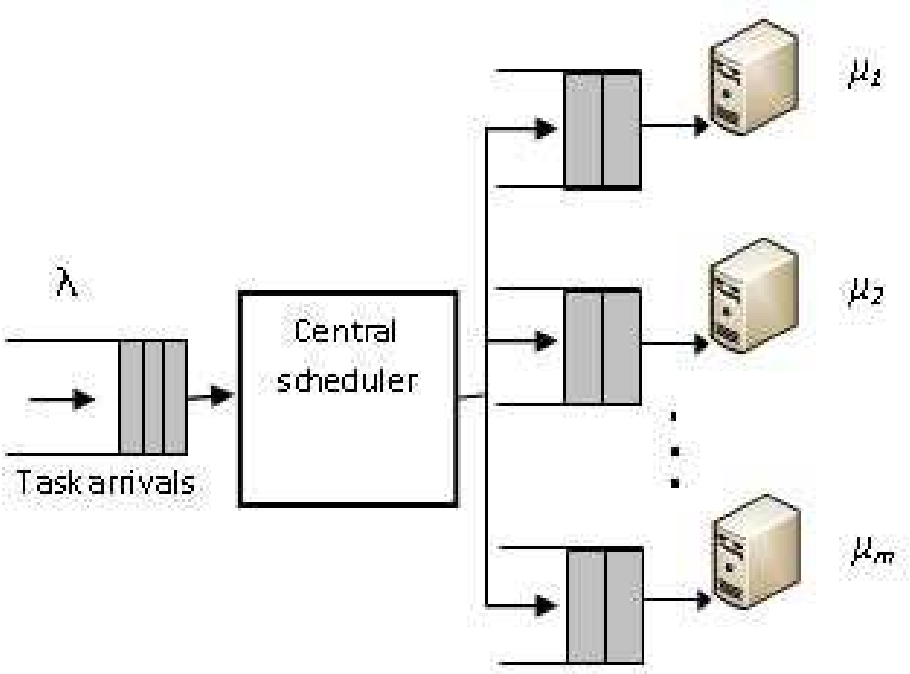

Figure 2.2: Heterogeneous Distributed Computing System with central scheduler

A centralized HDCS model can be characterized by the following:

- A finite collection of computing nodes and a waiting queue. The waiting queue can be accessed by all the computing units through a communication network.
- Every computing node can be evaluated by a time metric and designated with the exponential service time distribution.
- Tasks are arriving at the central scheduler following Poisson distribution with an arrival rate of λ.
- Allocation of a tasks to the computing nodes are equally probable and can be assigned by the central scheduler independently. It is assumed that if all the computing nodes are busy, the task will keep waiting in a waiting queue of infinite length at the central scheduler.

The tasks arriving from the different users to the central scheduler or serial scheduler have the same probability of being allocated to any one of the m computing nodes. Each computing node executes a single task at a time. The arrivals of the tasks at the central server or resource manager follow a Poisson process with an arrival rate of λ. Each of the computing nodes can be modeled as shown in Figure 2.2. The tasks that are to be executed at a node under the control of a local scheduler and the scheduling policy of the node is responsible for the execution of the assigned task. The centralized HDCS can be modeled as an $M/M/m$ (Markovian arrivals, Markovian service times, m computing nodes as server, and with infinite buffer for incoming task) multi-server queuing system.

2.4.2 Decentralized system model for HDCS

Decentralized dynamic load balancing algorithms are implemented in the individual computing nodes and operates on periodic information exchange between the computing nodes. A computing node in a decentralized HDCS is shown in Figure 2.3.

In a decentralized distributed system model the tasks are arriving independently at m computing nodes. Let tasks arrive at node M_j in accordance with a Poisson process with a rate λ_j. The total arrival rate to the system is denoted as $\lambda = \Sigma_{j=1}^{m} \lambda_j$. The tasks arriving at a computing node either serve at that node or as the result of load balancing decision, migrate to other computing nodes for execution.

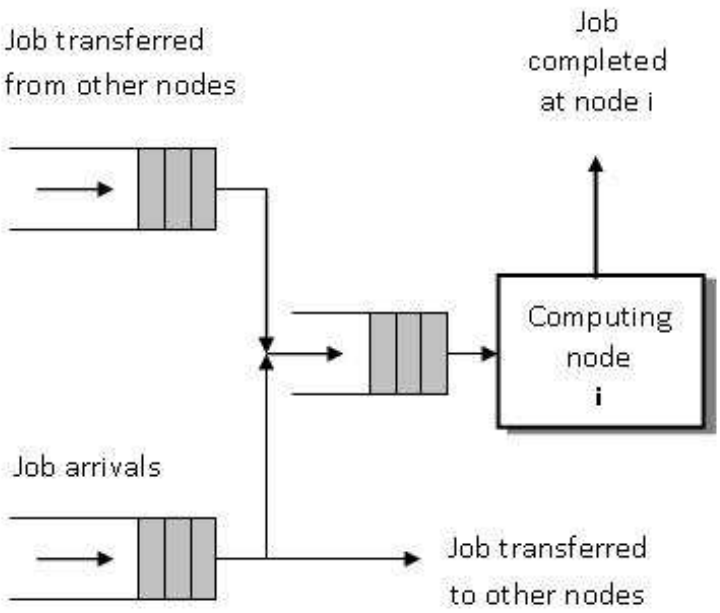

Figure 2.3: Computing Node in decentralized HDCS

2.5 Task or work load model

The workload submitted to the HDCS is assumed to be in the form of tasks. Depending on the dynamic scheduling approach, the tasks are submitted either to the central scheduler or submitted to different computing nodes independently. For different domains of computer science the exact meaning varies greatly. Terms such as application, task, sub task, job and program are used to denote the same object in some instances, and yet, have totally different meanings in others. We have assumed the task to be the computational unit to execute on the computing nodes of HDCS.

Definition 2.2 (Heterogeneous Distributed System task)**.** *A task is an independent scheduling entity and its execution cannot be preempted. The tasks are independent and can be executed in any node.*

Definition 2.3 (Meta-task)**.** *A meta-task is defined as a set of independent tasks with no data dependency.*

Formally, each arriving task t_i is associated with an arrival time and expected time to compute on different computing node. Let T be the set of task, $T = \{t_1, t_2, ..., t_n\}$. Each task t_i has an expected time to compute on node M_j, denoted as t_{ij}. Hence, the tasks are characterized by Expected Time to Compute (ETC) as in Table 2.1, where all m

Table 2.1: Expected Time to Compute: ETC

$Task/Node$	M_1	$\cdots$	M_j	$\cdots$	M_m
t_1	t_{11}	$\cdots$	t_{1j}	$\cdots$	t_{1m}
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$
t_i	t_{i1}	$\cdots$	t_{ij}	$\cdots$	t_{im}
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$
t_n	t_{n1}	$\cdots$	t_{nj}	$\cdots$	t_{nm}

computing nodes can be represented in the first row. In ETC matrix, the elements along a row indicate the execution time of a given task on different nodes [24]; in particular, t_{ij} represent expected time to compute i^{th} task on machine M_j.

A heterogeneous distributed computing system utilizes a distributed suite of different high-performance nodes, interconnected with high-speed links, to perform different computationally intensive applications that have diverse computational requirements. An application of distributed computing platform many be defined as:

Definition 2.4 (Application)**.** *A set of tasks that can be represented by a Directed Acyclic Graph (DAG), with operational precedence constraint among the task.*

Let a *meta-task* be represented by a set of n tasks. If the HDCS has m computing nodes, then tasks can be represented by an ETC matrix. But a DAG can be listed as a set of ordered tasks with level order traversal of the DAG. Hence, a program with n tasks can be represented as an $n \times m$ ETC matrix on m computing node. Hence, ETC matrix can be used to study dynamic load balancing problems in HDCS. Because there are no dependency among the tasks, load balancing schemes are simplified, and mostly focuses on efficient matching of tasks to the computing nodes [92]. It is assumed that the size of the *meta-task* is the number of tasks to be executed on the HDCS and is denoted as $|T| = n$. The major assumptions regarding the tasks to be executed on a HDCS are as follows:

- Tasks cannot be preempted once they begin to execute on a computing node.
- Only mapping heuristics can assign tasks to computing nodes, no task from external source is permitted.
- A task can be executed on one node of the system.

- When a nodes executing a task, there is no priority distinctions between the tasks with the computing node.
- A computing node cannot remain idle when the tasks are in waiting queue of the node.
- An estimation of the execution time for each task on each computing node is known a priori.

The ETC models presented in [24] are characterized by three parameters (i) machine heterogeneity, (ii) task heterogeneity and (iii) consistency. The task heterogeneity can be represented with two categories (i) consistent and (ii) inconsistent. A *consistent ETC matrix* can be obtained by arranging the computing nodes in order of their processing capability or may be arranged as decreasing order of FLOPS. In particular, if a node M_i has a lower execution time than node M_j for task t_k , then $t_{ki} < t_{kj}$. An *inconsistent ETC matrix* results in practice, when the HDCS includes different type of machine architectures such as high performance computing clusters, multi-core processor based workstations, parallel computers, and work stations with GPU units. In the literature most of the researchers assumed the *task execution times* to be uniformly distributed [24, 93, 94, 95]. The entire task has expected time to compute on m nodes of HDCS. Hence, the generalized load-balancing problem is to assign each task to one of the nodes M_j so that the loads placed on all of the nodes are as "balanced" as possible [86].

2.6 Dynamic load balancing as linear programming problem(LPP)

The dynamic load balancing problem of assigning n tasks on an HDCS with m computing nodes can be represented as an optimization problem to minimize the *makespan*. The tasks to be executed on the HDCS are represented by the ETC matrix and follows the basic assumptions as listed in Section 2.5. Let $A(j)$ be the set of tasks assigned to node M_j; and T_j be the total time machine M_j needs to finish all the task in $A(j)$. Hence $T_j = \sum_{t_i \in A(j)} t_{ij}$; for all task in $A(j)$. This is otherwise denoted as L_j and defined as load on node M_j. The basic objective of load balancing is to minimize the *makespan*, which is defined as maximum load on any node ($T = max_{j:1:m} T_j$). Let x_{ij} correspond to each pair (i, j) of node $M_j \in M$ and task $t_i \in T$ such that

$$x_{ij} = 0; \textit{ when the task } i \textit{ is not assign to node } M_j. \tag{2.1}$$

or

$$x_{ij} = t_{ij};\ when\ the\ task\ i\ is\ assigned\ to\ node\ M_j. \tag{2.2}$$

For each task t_i, we need $\sum_{j=1}^{m} x_{ij} = t_{ij}$; for all task $t_i \in T$. The load on node M_j can be represented as $L_j = \sum_{j=1}^{m} x_{ij}$, where x_{ij} is defined in Equations 2.1 and 2.2. The load balancing problem aims to find an assignment that minimizes the maximum load. Let L be the load of HDCS with m nodes. Hence, the generalized load balancing problem on an HDCS can be formulated as

$$Minimize\, L = \sum_{j=1}^{m} x_{ij} = t_{ij},\ \forall\, t_i \in T \tag{2.3}$$

subjected to:

$$\sum_{j=1}^{n} x_{ij} \leq L,\ \forall\, M_j \in M \tag{2.4}$$

$$where\ x_{ij} \in \{0, t_{ij}\}, \forall t_i\ \in T, and\ M_j \in M$$

$$x_{ij} = 0,\ \forall\, t_i \notin A(j)$$

The objective function 2.3 maps each possible solution of the load balancing problem to some non-negative value, and an optimal solution to the optimization problem is one that minimizes the value of this objective function. A feasible assignment is a one-to-one correspondence with x_{ij} satisfying the constraints in Equation 2.4. Hence, an optimal solution to this problem is the load L_j on the node M_j, also denoted as corresponding assignment $A(j)$. For n tasks to be assigned to m computing node, the number of possible allocation will be m^n and the number of states for execution will be $n!$. The *load balancing problem* is therefore intractable when the number tasks or computing nodes exceeds a few units.

The objective function defined here is to minimize the *makespan*, hence the *makespan* is used as the performance metric for evaluating various load balancing scheme through resource allocation. Moreover, the *makespan* has been used as the most common performance metric by a majority of researchers [6, 63, 95, 96, 97, 98].

2.7 Load balancing algorithm: State of the art

In dynamic resource allocation scenarios the responsibility for making global scheduling decisions are lie with one centralized schedular, or are shared by multiple distributed schedulers [54]. Hence, dynamic load balancing algorithms can be further classified into

a centralized approaches and a decentralized approaches. In a centralized approach [6, 55, 56] one node in the distributed system acts as the central controller and is responsible for task allocation to other computing nodes. The central controller takes the decision based on load information obtained from the other nodes in the distributed system. In a decentralized approach [39] all the computing nodes participated in the task allocation process. This decentralized decision making can be realized through a cooperation or without cooperation among the computing nodes.

The algorithms approaches used for the load balancing problem are roughly classified as (*i*) exact algorithms, (*ii*) heuristic algorithms, and (*iii*) approximation algorithm [87, 99]. An algorithmic approach to load balancing problem is presented in [86]. Different forms of linear programming formulation of the load balancing problem has been discussed along with *greedy*, *randomized* and *approximation algorithms* to produce sub-optimal solutions to the problem. The solution to this intractable problem was discussed under different algorithmic paradigms. The selection of load balancing algorithm mostly depends on the set of system parameters such as (i) system size, (ii) system load, and (iii) system traffic intensity [5].

Iterative load balancing methods rely on successive approximations to a global optimal work load distribution, and hence at each iteration, need be only to concerned with task migration to the computing nodes [38]. Xu and Lau [38] presented a classification of iterative dynamic load balancing strategies in multicomputer systems into two major group, (i) deterministic iterative strategies and (ii) stochastic iterative strategies.Three stochastic iterative strategies successfully used by the researchers to solve load balancing problem are: (i) randomized allocation, (ii) simulated annealing, and (iii) genetic algorithms. Heuristic algorithms can find approximate or sub-optimal solutions with acceptable time and space complexities, and are promising in solving intractable problems.

Algorithms where some of the actions are dependent on chance are generally termed as *probabilistic algorithms* or *randomized algorithms*. Randomization has long been used in algorithm design. Randomness can be used to find approximate numerical solution to problems having an exponential solution space [100]. Randomized algorithms are preferred over deterministic algorithm because: (i) they runs faster than the best known deterministic algorithm and (ii) they are simple to describe and implement than the deterministic algorithms. Four major subdivisions of randomized algorithms based upon uniqueness and correctness of solution are *numerical randomized algorithm*, *Monte-Carlo algorithm* , *Las Vegas algorithm* and *Sherwood algorithm*. In this thesis we have used randomized load balancing algorithms for comparative analysis in Chapter 3 and Chapter 4 using centralized scheduler.

Table 2.2: The Genetic Algorithm based load balancer in distributed system

GA Load balancer	Obj. function	Pop size	Selection	Crossover	Mutation	Heterogeneity	Load balancing
Zomaya et al.	maxspan	10	roulette wheel	NA	NA	Yes	Static
Subrata et al.	makespan	twice the task set	Tournament	0.8	0.0005	Yes	Dynamic
Kumar et al.	makespan	20	roulette wheel	0.8	0.2	No	Dynamic
kolodziej et al.	flow time	NA	Liner ranking	0.9	0.4	Yes	Dynamic
Greene et al.	time duration	20	roulette wheel	NA	NA	Yes	Dynamic
Page et al.	makespan	20	roulette wheel	NA	NA	Yes	Dynamic
Aguilar et al.	cost function	NA	roulette wheel	NA	NA	Yes	Static
Braun et al.	makespan	200	roulette wheel	0.6	0.4	Yes	Static
Lee and Hwang	CPU queue length	50	wheel of fortune	0.7	0.05	No	Dynamic
Nikravan et al.	CPU utilization	50	roulette wheel	0.9	0.1	No	Static

Simulated annealing is a general and powerful iterative technique for combinatorial optimization problems.The technique is Monte Carlo in nature, which simulates the random movements of a collection of vibrating atoms in the process of cooling [101]. The dynamic load balancing algorithm using simulated annealing exercise are initiated from the central scheduler in an HDCS. The global workloads are the tasks waiting with central scheduler to be allocated to the computing nodes after a fixed number of iterations.

Genetic algorithms are meta-heuristics based on the iterative application of stochastic operators on a population of candidate solutions [102]. They are proved to be useful heuristic approaches to find sub-optimal solutions for the problems with an exponential solution space [103, 104]. In the process of problem solving in genetic algorithm, solutions are selected from the population in each iteration. The selected solutions are subjected to recombination with genetic operators to produce new solutions. These solutions may replace other solutions selected randomly or through a selection strategy suitable to the problem domain. A typical genetic algorithms are characterized by following attributes; the genetic representation of candidate solutions, the population size, the evaluation function, the genetic operators, the selection algorithm, cross over probability, mutation probability, the generation gap, and the amount of elitism used. However based on the various researchers finding Table 2.2 presents a comparison of various genetic algorithm based schedulers. The NA indicates the non availability of the information relation to genetic scheduler.

The limitation of integer programming tools(exact algorithm) is that, it does not provide any guarantee to produce a quality of solution in reasonable running time. Moreover, the heuristic methods are also suffers from certain drawbacks. In particular it requires a well defined analysis to evaluate the quality of heuristics besides the excellent numerical performance. The approximation algorithm addresses both the issue of guarantee and making feasible solution. Also an approximation algorithm is polynomially bounded and characterised approximation ratio[105].

The central or serial scheduler schedules the processes in a distributed system to make use of the system resources in such a manner that resource usage, response time, network congestion, and scheduling overhead are optimized.

In a decentralized approach [39] all the computing nodes participated in the task allocation process. This decentralized decision making can be realized through a cooperation or without cooperation among the computing nodes. Most of the decentralized approaches use the partial information available with the individual computing nodes to make sub-optimal decisions. The scope of applying game theoretic techniques to load balancing in distributed computer systems has been analyzed in the context of Nash

equilibrium by a majority of the researchers. To facilitate a game theoretic approach, the HDCS is viewed as the collection of computing resources that are under the supervision of the server with each node. The *scheduler* or *load balancer* is available as a component of the server to facilitate the task allocation. This chapter presents a non-cooperative game theoretic framework for dynamic load balancing in heterogeneous distributed systems with the goal of achieving Nash equilibrium.

2.8 Conclusion

This chapter introduced the basic concepts, the terminology and the state of the art of dynamic load balancing in heterogeneous distributed computing systems. Dynamic load balancing problem on an HDCS is represented as a linear programming problem, with the objective of minimizing the *makespan*. A model for a Heterogeneous Distributed Computing System(HDCS) has been presented. It includes the system architecture and workload model that has to be followed for the design of dynamic load balancing algorithms for the HDCS. To this end, the basic concepts of the load balancing algorithm theory were over viewed, focusing primarily on the dynamic load balancing of tasks on heterogeneous computing nodes.

Chapter 3

Impact of System Heterogeneity with Greedy Resource Allocation Algorithms

This chapter presents experiments with greedy resource allocation algorithms using batch mode paradigms and its approach to find an optimal or sub-optimal solutions for the dynamic load balancing problem with attempts to minimize the makespan on an HDCS. The simulation result in this chapter show that the greedy based scheduling policy depends on system heterogeneity. The different types of heterogeneity in an HDCS is represented as consistent and inconsistent ETC matrix models. The relative performance of the heuristics under different circumstances has been simulated on four different HDCSs. Simulation study has been presented to determine the impact of a simple heuristic on task allocation policies in the heterogeneous distributed system.

3.1 Introduction

Dynamic load balancing strategies are categorized into centralized or distributed controlled strategies. The system state changes with time on arrival of tasks from the user. The effectiveness of any load balancing scheme depends on the quality of load measurement and prediction that indicate the degree of load imbalance in the system [36]. In dynamic load balancing, the decision to allocate tasks are taken on the fly considering the load on different computing nodes during task execution. The greedy heuristic constructs a feasible solution from scratch and defines the mapping of tasks to the computing

nodes. In these algorithms tasks are sorted with a given criteria, and then mapped in that order to the computing nodes in the HDCS. The allocation of one task depends on previously allocated tasks. Greedy algorithms maintain feasibility while generating the optimal solution by considering the resource constraints [84]. The heuristics based scheduling algorithms used for load balancing can be grouped into two category: *on-line mode* and *batch mode* [8, 53, 96, 97, 106, 107, 108]. In the *on-line mode* (also known as the immediate mode) a task is mapped onto a computing node as soon as it arrives at the scheduler. In *batch mode* heuristics, the tasks are not mapped onto the machines as they arrive; instead they are collected in a set that is examined for mapping at pre-scheduled time [96]. The batch mode scheduling is the most appropriate framework to design dynamic load balancing algorithms. This is a simple scheduling scenario in an HDCS, and is useful to illustrate many real-life approaches that utilizes parallel nature of the HDCS, enabling independent computation of tasks on the nodes [109].

The greedy paradigm provides a framework to design an algorithm, that works in stages, considering one input at a time. At each stage a particular input is selected through a selection procedure. Then a decision is made regarding the selected input, whether to include it into the partially constructed optimal solutionor not [110]. The selection procedure can be realized in $\bigcirc(1)$ a time with the use of a binary heap (max heap or min heap) data structure, with time complexity of $\bigcirc(log\ n)$ for an instance of size n. Hence, the realization of a greedy heuristic is the simplest and the selection procedure can be realized with worst case time complexity of $\bigcirc(log\ n)$. All the load balancing algorithms discussed in this chapter uses a selection procedure with best case time complexity of $\bigcirc(1)$ and worst case time complexity of $\bigcirc(log\ n)$. The greedy heuristic algorithms presented in this chapter are simulated to study the load balancing in four different HDCSs by varying the system heterogeneity. The simple heuristic algorithms are the First-Come, First-Served (FCFS) algorithm that follows the order of arrival time of the task with the central scheduler, the second algorithm (MINMIN) selects the task with minimum ETC on the node to be allocated, the third algorithm (MINMAX)selects the task with maximum ETC on the node to be allocated and the fourth one is the random task allocation algorithm that selects the node randomly from m nodes to allocate task t_i.

The load balancing in distributed computing systems becomes a major research issue to utilize the ideal computing resources. This chapter presents the centralized dynamic load balancing algorithm, that operates in batch mode, to realize the concept of dynamic allocation. A node operates as the central scheduler and collects the load information from the other computing nodes in the HDCS to finalize the allocation decision.

3.2 Related work

Centralized static resource allocation algorithms have been studied extensively in [111, 92] using heuristics. Tasks are assigned to the computing nodes with a centralized load balancing algorithm, a central node collecting the load information from the other computing nodes in the HDCS. Dandamudi [112] has presented a study on impact of heterogeneity and variance in inter-arrival times in an HDCS with two different arrival rates of the tasks to central scheduler. However, he has not considered the task model and expected time to execute a task on different systems. Most scheduling heuristics used in HDCSs try to minimize the *makespan*. Tseng et al.[58] presented a simulation study to compare five different scheduling heuristics to minimize the *makespan* in a dynamic environment for grid computing systems. A quality of service guided new *min-min* algorithm based on a general adaptive scheduling environment has been presented by Xiaoshan *et al.* in [96]. Izakian *et al.* [97] have presented an efficient heuristic method for scheduling independent tasks on heterogeneous distributed environments and compare it with five popular heuristics for minimizing the *makespan*. The paper by Xhafa and Abraham [98] reveals the complexity of the scheduling problem in computational grids when compared with classical parallel and distributed systems. A static mapping of *meta-tasks* to minimize the total execution time has been presented by Braun *et al.* [95] for an HDCS. Batch mode scheduling using *min-min, max-min, and the surffrage* heuristics has been demonstrated in [58, 113, 114, 115, 116, 117, 118] with *makespan* as the performance parameter.

3.3 Greedy load balancing algorithm

Using the *makespan* as the load balancing metric for a given distributed computing environment should reasonably predict the performance of the system [53]. Makespan is also used in the mathematical model of the load balancing problem discussed in algorithm design [86]. This chapter follows a simple *Greedy-Balance* algorithmic framework, discussed by Kleinberg and Tardos [86] to suggest a generalized greedy load balancing algorithm for an HDCS. The proposed algorithms operates with the *ETC matrix* and *arrival time* for each task, to allocate the task to computing nodes for load balancing. The arrival time of the task is to be recorded in a priority queue $HAT(MaxTask)$. The priority queue, implemented as *min-heap*, records the order at which the tasks are arriving at the central scheduler. The *min-heap* can be created with a time complexity of $\bigcirc(log\ n)$. The task with the earliest arrival time is selected and assigned to the machine with the minimum load. Further, it is assumed that the initial load of each of the computing

node is zero. The greedy Algorithm 3.1 is designed to obtain an optimal task assignment by assigning the n tasks in stages, one task per stage in a non-decreasing order of task arrival time. The greedy load balancing algorithm operates by initializing the set of task $A(j)$ and the total time to finish the task T_j for every node M_j. Algorithm 3.1 can be implemented with time complexity $\bigcirc(n\ log\ n)$. This algorithm successfully terminates when task queue becomes empty. The selection of the computing node is based upon a greedy criterion : *assign the task to the node with minimum* T_j. The algorithm computes *makespan* for the set of task having $MaxTask$ number of tasks.

Algorithm 3.1 `Greedy load balancing algorithm with priority queue`

Require: $ETC(MaxTask, MaxNode), HAT(MaxTask) : task\ Queue$
Ensure: $L : makespan$
1: $L_j \longleftarrow 0\ forall$ node M_j
2: $A(j) \longleftarrow \phi\ forall$ node M_j, Let ϕ be the empty set
3: **repeat**
4: Let M_j be a node with minimum T_j
5: Let t_i be the task on root of the min-heap HAT
6: Allocate task t_i to Node M_j
7: $A(j) \longleftarrow A(j) \cup \{t_i\}$
8: $L_j \longleftarrow L_j + t_{ij}$
9: Remove task t_i from min-heap HAT
10: **until** HAT is not empty
11: $L \longleftarrow max_j L_j$

The objective of a dynamic load balancing algorithm is to allocate the tasks *on the fly* as the tasks arriving according to a Poisson process are queued with the central scheduler. A fixed batch size, denoted as $WinSize$, represents the number of tasks selected in a batch for allocation. As there are too many tasks waiting to be allocated with the central scheduler, the scheduling heuristics are applied only to the tasks that are within the batch. The greedy heuristic algorithms listed in Section 3.6 are being used to allocate the tasks for load balancing. Once one batch of tasks are allocated to the computing nodes, the next batch of tasks are selected from the task queue for allocation. Algorithm 3.1 can be modified to Algorithm 3.2 with $MaxTask = WinSize$ to facilitate batch mode resource allocation. Algorithm 3.2 can be called $\frac{n}{WinSize}$ times in batch mode to allocate n tasks dynamically to m computing node. For simplicity it is assumed that $MaxTask$ is an integer multiple of $WinSize$. Algorithm 3.2 is executed for the first time with the following initialization:

$$L_j \longleftarrow 0\ for\ all\ node\ M_j$$

$$A(j) \longleftarrow \phi \; for \; all \; node \; M_j$$

The *makespan* can be obtained after $\frac{n}{WinSize}$ steps as $L = max_j L_j$.

Algorithm 3.2 `Greedy load balancing for batch job with priority queue`

Require: $T, A, ETC(WinSize, MaxNode), HAT(WinSize) : task\ Queue$

1: **repeat**
2: Let M_j be a node with minimum L_j
3: Let t_i be the task on root of the min-heap HAT
4: Allocate task t_i to Node M_j
5: $A(j) \longleftarrow A(j) \cup \{t_i\}$
6: $L_j \longleftarrow L_j + t_{ij}$
7: Remove task t_i from min-heap HAT
8: **until** HAT is not empty

3.4 System and task heterogeneity

An HDCS model consists of n heterogeneous computing nodes, which represent host of computers having different processing abilities, connected by an underlying communication network [37]. For convenience, we use *node*, and *computing node* interchangeably in the rest of this thesis. The assumptions on *computing node* and *task* are discussed in Section 2.3 and 2.5 respectively. The HDCS can be characterized by using the expected time to execute the task on different computing nodes present in the system. Hence, the heterogeneity of the system is defined as:

Definition 3.1 (Machine heterogeneity)**.** *Machine heterogeneity, otherwise known as computing node heterogeneity, is the variation among the execution times for a task on all the machine in the HDCS.*

Definition 3.2 (Task heterogeneity)**.** *Task heterogeneity is defined as the amount of variance among the execution times of the tasks in the meta-task for a given machine.*

A task has different execution times if it executed on different heterogeneous computing nodes of an HDCS. The expected execution time of task t_i is denoted as t_{ij} when assigned to node M_j without any load. The completion time C_{ij} of task t_i on node M_j is defined as the wall-clock time at which the node completes the task t_i, and is computed as $C_{ij} = t_{ij} + L_j$, where L_j is the load of the node M_j when task is assigned. Execution times of the tasks on different computing node are estimated and represented using the Expected Time to Compute (ETC) matrix model in [24]. We have consistent

Algorithm 3.3 `Range based ETC generation algorithm`

Require: $R_{task}, R_{mach}, MaxTask, MaxNode$
Ensure: ETC
1: **for** $i = 1$ to MaxTask **do**
2: $x(i) \longleftarrow 1 + (R_{task} - 1) * rand(1)$
3: **for** $j = 1$ to MaxNode **do**
4: $ETC(i, j) \longleftarrow round(x(i) * (1 + (R_{mach} - 1) * rand(1)))$
5: **end for**
6: **end for**

as well as the inconsistent ETC matrix to demonstrate the resource allocation abilities of four greedy algorithms. To generate the ETC matrix, we have used the range based ETC generation technique suggested in [24]. The ETC generation process is outlined in Algorithm 3.3. Let R_{task} and R_{mach} be the numbers representing task heterogeneity and machine heterogeneity respectively. In this chapter we have used range based ETC generation algorithm with the typical value for R_{task} and R_{mach} as follows:

- R_{task} is 10^5 and 10 for high and low heterogeneity respectively.
- R_{mach} is 10^2 and 10 for high and low heterogeneity respectively.

The ETC matrix for simulation are generated by using two uniform distribution $U(1, R_{task})$ and $U(1, R_{mach})$ and are realized as:

$$1 + (R_{task} - 1) * rand(1)$$

and

$$1 + (R_{mach} - 1) * rand(1)$$

where $rand()$ function generates a value between $(0, 1)$.

The ETC generation Algorithm 3.3 uses $R_{task} = 1000$ and $R_{mach} = 50$ respectively. We have assumed that the expected time to compute the task t_i on node M_j is the integer values in time unit of seconds. An example of inconsistent ETC matrix generated for 15 tasks on 7 nodes is shown in Table 3.1. If the computing nodes are arranged in the decreasing order of their processing rate, then a consistent ETC matrix results. The example of a consistence ETC matrix generated for 15 tasks on 7 nodes using Algorithm 3.3 is shown in Table 3.2.

Table 3.1: Inconsistent ETC matrix for 15 task on 7 nodes

	M_1	M_2	M_3	M_4	M_5	M_6	M_7
t_1	2610	67	2113	2226	2362	279	1116
t_2	5254	2893	5962	1295	1820	1063	1002
t_3	12785	12149	3523	18608	13687	7911	11369
t_4	952	2570	1419	2017	2571	4321	692
t_5	21351	11310	11274	7922	20362	8620	2911
t_6	7845	5013	8116	2235	2915	18409	18678
t_7	1132	3602	5272	11878	505	895	2673
t_8	11974	10637	7504	9034	5042	12181	3333
t_9	3434	6549	10880	13484	1710	15996	13408
t_{10}	5453	5584	3905	6321	6348	10016	9743
t_{11}	6306	13148	8743	5865	15162	14165	9017
t_{12}	9275	3484	4911	7502	3831	13203	3285
t_{13}	1065	1383	2542	1848	5260	2511	1144
t_{14}	22178	10184	2917	6175	9516	13644	6267
t_{15}	10820	3581	2038	4689	5016	6575	7813

Table 3.2: Consistent ETC matrix for 15 task on 7 nodes

	M_1	M_2	M_3	M_4	M_5	M_6	M_7
t_1	1121	2379	10195	12843	13673	14864	14946
t_2	1162	1463	1884	2306	2524	2872	3380
t_3	2899	3284	7476	9005	11566	16452	20826
t_4	2282	3486	3618	4076	4145	6782	7298
t_5	2764	8164	9676	21642	21777	22372	30581
t_6	340	3061	4569	8618	8712	10029	10828
t_7	637	12022	13132	14212	14257	23342	27648
t_8	889	2741	3108	7787	7820	11752	12749
t_9	3553	4409	4848	10514	12709	13162	15706
t_{10}	1035	2859	4142	5264	5539	6173	6534
t_{11}	4439	7282	8148	20835	22207	23168	30658
t_{12}	11036	13000	14012	14267	20319	20320	23545
t_{13}	11423	11767	23154	24867	26155	35210	36738
t_{14}	2181	7808	8827	10304	10402	10429	14998
t_{15}	2240	3366	3470	6377	14481	19777	27049

3.5 Queueing model for load balancing

The load balancing problem has been evenly treated, in both the fields of computer science and operations research. Queueing models are used as the key model for performance analysis and optimization of parallel and distributed systems [95]. Scheduling of tasks while balancing the load is a distributed system involves deciding not only when to execute a process, but also where to execute it. Accordingly, scheduling in a distributed system is accomplished by two components: *the allocator* and *the scheduler*. The allocator decides where a job will execute and the scheduler decides when a job gets its share of the computing resource at the node to which it is allocated. Queueing models can be viewed as key models for the performance analysis and optimization of parallel and distributed systems [48]. A queueing theoretic approach was used to analyse the performance of heterogeneous multiprocessor computer system involving random environments can be found in [119]. Optimal load balancing strategies are modeled using the queueing theory by Spies [29] who obtained analytical results through simulation. Rykov and Efrosinin [120] have presented an algorithm to find optimal threshold levels for different queueing systems with heterogeneous servers. Wang *et al*. [121] has presented a study on *maximum likelihood estimates* as well as *confidence intervals* of an $M/M/m$ queue with heterogeneous servers under steady-state conditions. An analysis using a queueing system with two heterogeneous server and a threshold type queue discipline is presented in [122]. The performance characteristics for a queueing system with heterogeneous servers has been presented with the calculation of the steady-state probabilities and waiting time by Vladimir *et al*. [123].

An HDCS can be modeled as a $M/M/m$ queueing system with heterogeneous server as discussed in [29, 121, 122] . The heterogeneous distributed computing system addressed in this work can be expressed by using a Kendall notation [91] or as like M/M/m, where: (i) First M: represents exponential inter arrival times between jobs(tasks) distribution, (ii) Second M: represents exponential execution time of jobs, and (iii) m: represents number of heterogeneous computing nodes in the system. It is also assumed that the queue has infinite buffer to accept the incoming tasks. Each computing node executes its queue of tasks in a first-come first-served order. Let the task enter into the queue at the central scheduler at a mean rate, λ. The distribution is assumed to be exponential with mean $\frac{1}{\lambda}$. A task t_i with the central scheduler can be allocated to a computing node with a probability a_i; hence, $\sum_{i=1}^{m} a_i = 1$. The processing time of task t_i on node M_j is modeled as an independent exponentially distributed random variable with mean $\frac{1}{\mu_j}$ [124]. For stability, it is also assumed that tasks must not be generated faster than the

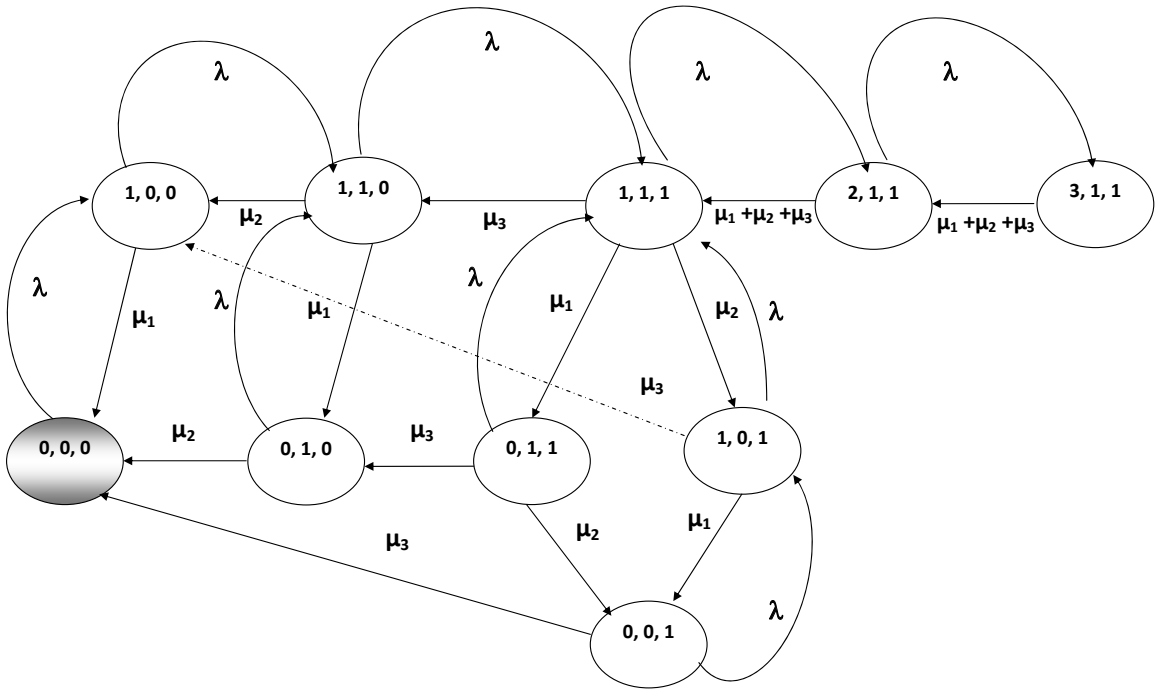

Figure 3.1: The state diagram for M/M/3 non-preemptive queue serving five task

rate at which the HDCS can process i.e. $\lambda \leq \sum_{j=1}^{m} \mu_j$.

Let us consider there are three heterogeneous computing nodes with five task to execute. This can be modeled as an M/M/3 queue with infinite buffers. Each heterogeneous computing node is multitasking and can accommodate a finite number of tasks assigned by the central scheduler. This queue can be analyzed by drawing a state transition diagram with a state represented by $m = 3$ tuple and denoted as $p(s_1, s_2, \ ... \ s_m)$. The Markov chain shown in Figure 3.1 describes the behaviour of the central scheduler and also explains the task migration phenomenon before the task begins execution.

Figure 3.1 represents the state diagram for the heterogeneous computing node $M_1,\ M_2,\ M_3$ with unequal mean service rates $\mu_1,\ \mu_2,\ \mu_3$ where $\mu_1 > \mu_2 > \mu_3$. It is assumed that the central scheduler runs on M_1. When more than one task is with the central scheduler, it

assigns the tasks to the other nodes. The state of the system is defined to be the tuple (s_1, s_2, s_3) where $(s_1 \geq 0)$ denotes the number of tasks in the queue including the task on execution with node M_1, and $s_2, s_3 \in \{0, 1\}$ denotes number of task at the nodes M_2 and M_3 respectively. Tasks arriving from the users have been modeled as a Poisson process with an arrival rate λ and queued at the central scheduler. When all the three nodes are idle, the faster computing node M_1 is scheduled for the task execution before the slow nodes M_2, M_3. Let $p(s_1, s_2, s_3)$ denotes the probability of the system state with s_1, s_2, and s_3 number of tasks scheduled to the three nodes. In steady state the following equations are used:

- $p(0, 0, 0)$: probability that there are no task in the system, otherwise denoted as p_0.
- $p(1, 0, 0)$: probability that there are one task in the system with M_1.
- $p(0, 1, 0)$: probability that there are one task in the system with M_2.
- $p(0, 0, 1)$: probability that there are one task in the system with M_3.
- p_1 denotes the probability that there is a single task and defined as: $p_1 = p(1, 0, 0) + p(0, 1, 0) + p(0, 0, 1)$
- $p(1, 1, 0)$: probability that there are two task in the system with first and second node.
- p_2 denotes the probability that there are two tasks and defined as $p_2 = p(1, 1, 0) + p(1, 0, 1) + p(0, 1, 1)$.
- p_3 denotes the probability that there are three tasks in the system and defined as $p_3 = p(1, 1, 1)$.
- p_4 denotes the probability that there are four tasks in the system and defined as $p_4 = p(2, 1, 1)$.
- p_5 denotes the probability that there are five tasks in the system and defined as $p_5 = p(3, 1, 1)$.

Steady-state equations for an M/M/3 queue with three heterogeneous servers are given by:

$$\lambda p(0, 0, 0) = \mu_1\ p(1, 0, 0) + \mu_2\ p(0, 1, 0) + \mu_3\ p(0, 0, 1) \tag{3.1}$$

$$(\lambda + \mu_1)p(1, 0, 0) = \mu_2\ p(1, 1, 0) + \mu_3\ p(1, 0, 1) + \lambda\ p(0, 0, 0) \tag{3.2}$$

$$(\lambda + \mu_2)p(0,1,0) = \mu_1\ p(1,1,0) + \mu_3\ p(0,1,1) \tag{3.3}$$

$$(\lambda + \mu_3)p(0,0,1) = \mu_1\ p(1,0,1) + \mu_2\ p(0,1,1) \tag{3.4}$$

$$(\lambda + \mu_1 + \mu_2)p(1,1,0) = \mu_3 p(1,1,1) + \lambda\ p(1,0,0) + \lambda\ p(0,1,0) \tag{3.5}$$

$$(\lambda + \mu_2 + \mu_3)p(0,1,1) = \mu_1 p(1,1,1) \tag{3.6}$$

$$(\lambda + \mu_1 + \mu_3)p(1,0,1) = \mu_2 p(1,1,1) + \lambda\ p(0,0,1) \tag{3.7}$$

$$(\lambda+\mu_1+\mu_2+\mu_3)p(1,1,1) = (\mu_1+\mu_2+\mu_3)p(2,1,1)+\lambda\ p(0,1,1)+\lambda\ p(1,0,1)+\lambda\ p(1,1,0) \tag{3.8}$$

$$(\lambda + \mu_1 + \mu_2 + \mu_3)p(2,1,1) = (\mu_1 + \mu_2 + \mu_3)p(3,1,1) + \lambda\ p(1,1,1) \tag{3.9}$$

As p_1 has three possibilities $p(1,0,0)$ or $p(0,1,0)$ or $p(0,0,1)$,it can be obtained from Equation 3.1 as,

$$p_1 \ = \ \frac{\lambda p_0}{\mu_1} \ + \ \frac{\lambda p_0}{\mu_2} \ + \frac{\lambda p_0}{\mu_3} \tag{3.10}$$

$$or, \qquad p_1 \ = \ \lambda p_0 \left(\frac{\mu_1\mu_2 + \mu_2\mu_3 + \mu_1\mu_3}{\mu_1\mu_2\mu_3} \right) \tag{3.11}$$

Similarly, p_2 can be computed by adding Equations 3.2, 3.3 and 3.4.

$$\lambda\left[p(1,0,0) + p(0,1,0) + p(0,0,1)\right] \ + \ \mu_1\ p(1,0,0) + \mu_2\ p(0,1,0) + \mu_3\ p(0,0,1) \ =$$

$$(\mu_1 + \mu_2)p(1,1,0) + (\mu_1 + \mu_3)p(1,0,1) + (\mu_2 + \mu_3)p(0,1,1) \ + \ \lambda p(0,0,0)$$

Using Equation 3.1

$$\lambda\left(p(1,0,0) + p(0,1,0) + p(0,0,1)\right) + \lambda p(0,0,0) \ = \ (\mu_1+\mu_2)p(1,1,0)+(\mu_1+\mu_3)p(1,0,1)+$$

$$(\mu_2 + \mu_3)p(0,1,1) \ + \ \lambda p(0,0,0)$$

On cancelling $\lambda p(0,0,0)$ from both LHS and RHS, we have

$$\lambda p_1 \ = \ (\mu_1 + \mu_2)p(1,1,0) + (\mu_1 + \mu_3)p(1,0,1) + (\mu_2 + \mu_3)p(0,1,1) \tag{3.12}$$

The system can have two tasks with three possibilities $p(1,1,0)$ or $p(0,1,1)$ or $p(1,0,1)$. So, p_2 can be obtained as follows:

$$p_2 = \frac{\lambda p_1}{\mu_1 + \mu_2} \ + \frac{\lambda p_1}{\mu_2 + \mu_3} + \frac{\lambda p_1}{\mu_1 + \mu_3}$$

or,

$$p_2 = \lambda p_1 \left(\frac{1}{\mu_1 + \mu_2} + \frac{1}{\mu_2 + \mu_3} + \frac{1}{\mu_1 + \mu_3} \right) \tag{3.13}$$

The system can have three tasks with only one possibility $p(1,1,1)$. So, p_3 can be computed by adding Equations 3.5, 3.6 and 3.7 as follows:

$$(\lambda + \mu_1 + \mu_2)p(1,1,0) + (\lambda + \mu_2 + \mu_3)p(0,1,1) + (\lambda + \mu_1 + \mu_3)p(1,0,1) =$$

$$\mu_3 p(1,1,1) + \lambda\ p(1,0,0) + \lambda\ p(0,1,0) + \mu_1 p(1,1,1) + \mu_2 p(1,1,1) + \lambda\ p(0,0,1)$$

or,

$$\lambda\left[p(1,1,0) + p(0,1,1) + p(1,0,1)\right] + (\mu_1 + \mu_2)p(1,1,0) + (\mu_1 + \mu_3)p(1,0,1) +$$

$$(\mu_2 + \mu_3)p(0,1,1) = (\mu_1\ +\ \mu_2\ +\ \mu_3)p(1,1,1) + \lambda\left(p(1,0,0) + p(0,1,0) + p(0,0,1)\right)$$

Using Equation 3.12

$$\lambda\left[p(1,1,0) + p(0,1,1) + p(1,0,1)\right] + \lambda p_1 =$$

$$(\mu_1\ +\ \mu_2\ +\ \mu_3)p(1,1,1) + \lambda\left(p(1,0,0) + p(0,1,0) + p(0,0,1)\right)$$

As p_2 has three possibilities $p(1,1,0)$ or $p(0,1,1)$ or $p(1,0,1)$, above equation can be simplified to

$$\lambda p_2 + \lambda p_1 = (\mu_1\ +\ \mu_2\ +\ \mu_3)p_3 + \lambda\left(p(1,0,0) + p(0,1,0) + p(0,0,1)\right)$$

Similarly p_1 is possible through $p(1,0,0)$ *or* $p(0,1,0)$ *or* $p(0,0,1)$, hence the above is further simplified as,

$$\lambda p_2 + \lambda p_1 = (\mu_1\ +\ \mu_2\ +\ \mu_3)p_3 + \lambda p_1$$

or,

$$\lambda p_2 = (\mu_1\ +\ \mu_2\ +\ \mu_3)p_3$$

or,

$$p_3 = \frac{\lambda p_2}{\mu_1\ +\ \mu_2\ +\ \mu_3} \tag{3.14}$$

The system can have four tasks with one possibility $p(2,1,1)$. So, p_4 can be computed from Equations 3.8 as,

$$(\mu_1 + \mu_2 + \mu_3)p(2,1,1) =$$

$$\lambda p(1,1,1) + (\mu_1 + \mu_2 + \mu_3)p(1,1,1) - [\lambda\ p(0,1,1) + \lambda\ p(1,0,1) + \lambda\ p(1,1,0)]$$

As p_2 has three possibilities $p(1,1,0)$ or $p(0,1,1)$ or $p(1,0,1)$, we have

$$(\mu_1+\mu_2+\mu_3)p(2,1,1) = \lambda p(1,1,1) + (\mu_1+\mu_2+\mu_3)p(1,1,1) - \lambda\ p_2$$

or,

$$p(2,1,1) = \frac{\lambda p(1,1,1)}{(\mu_1+\mu_2+\mu_3)} + p(1,1,1) - \frac{\lambda\ p_2}{(\mu_1+\mu_2+\mu_3)}$$

or,

$$p_4 = \frac{\lambda p_3}{(\mu_1+\mu_2+\mu_3)} + p_3 - \frac{\lambda\ p_2}{(\mu_1+\mu_2+\mu_3)}$$

Using Equation 3.14

$$p_4 = \frac{\lambda p_3}{(\mu_1+\mu_2+\mu_3)} + p_3 - p_3$$

or,

$$p_4 = \frac{\lambda p_3}{(\mu_1+\mu_2+\mu_3)} \tag{3.15}$$

Similarly p_5 can be obtained from Equation 3.9 as,

$$(\mu_1+\mu_2+\mu_3)p(3,1,1) = (\lambda+\mu_1+\mu_2+\mu_3)p_4 - \lambda\ p(1,1,1)$$

Using Equation 3.15

$$p_5 = \frac{\lambda p_4}{(\mu_1+\mu_2+\mu_3)} + p_4 - \frac{\lambda\ p_3}{(\mu_1+\mu_2+\mu_3)}$$

or,

$$p_5 = \frac{\lambda p_4}{(\mu_1+\mu_2+\mu_3)} \tag{3.16}$$

Solving recursively, analytic solutions for probability that there are n number of tasks in the HDCS with m computing nodes is denoted as p_n or $p(n-2,1,1)$ and is derived as follows:

$$p_n = \frac{\lambda p_{n-1}}{\mu_1+\mu_2+\cdots+\mu_m} \tag{3.17}$$

With three heterogeneous computing nodes, the traffic intensity for this system can be computed as

$$\rho\ =\ \frac{\lambda}{\mu_1\ +\ \mu_2\ +\ \mu_3}$$

Since the system is in a steady state, so $\rho\ <\ 1$, or equivalently $\lambda\ <\ \mu_1\ +\ \mu_2\ +\ \mu_3$. Similarly for an HDCS with m computing nodes

$$\rho\ =\ \frac{\lambda}{\mu_1\ +\ \mu_2\ \cdots+\ \mu_m}$$

In general for the HDCS with m nodes, the state of the markov chain is described by the m tuple $(s_1, s_2, \cdots, s_m)$ in which s_j denotes the number of tasks with node M_j. A task is allocated to node M_j is with a probability, a_j. Let λ_j be the arrival rate of tasks at the computing node M_j due to allocation by the central scheduler. The average utilization ρ_j can be computed as $\frac{\lambda_j}{\mu_j}$. Let Q_j be the queue length of node M_j. Then the average queue length can be computed as,

$$E(Q_j) = \frac{\rho_j}{1 - \rho_j}$$

The average response time, denoted as $E(T_j)$ is defined as

$$E(T_j) = \frac{1}{\lambda}\left(\frac{\rho_j}{1 - \rho_j}\right)$$

As the central scheduler runs on M_1 , let a_1 is the probability that the task is scheduled to node M_1 locally. The probability that a task will migrate to another node is $1 - a_1$ and migration probabilities to all the nodes are identical. The *average execution queue length* L_j, otherwise known as the *load* on node M_j, determines how smoothly the load is balanced.

3.6 Greedy heuristic algorithms for load balancing

Heuristic and meta-heuristic algorithms are the effective strategies for scheduling in an HDCS due to their ability to deliver high quality solutions in reasonable time [57]. In this section, we present the greedy algorithms for task allocation in the HDCS. The heuristics used are very simple to realize with very little computational cost in comparison to the effort by resource allocation algorithms. A randomized resource allocation algorithm is selected along with the heuristic algorithms because the randomness can guarantee average case behaviour as well as it produces efficient approximate solutions to intractable problems. The dynamic load balancing algorithms using batch mode heuristics MINMIN and MINMAX operate by selecting a fixed small number that fits to the task window on each iteration. The MINMIN and MINMAX operate for a fixed number of iterations to assign n tasks to the computing nodes.

3.6.1 First-Come, First-Served (FCFS) heuristic

The FCFS heuristics is a very simple and most common resource allocation heuristic being used by various researcher to study task scheduling in distributed system [6, 92, 95, 112].

This is a non-preemptive scheduling policy that schedules tasks in the order of their arrival to the central scheduler. The FCFS algorithm, i.e., Algorithm 3.4 is applied to the load balancing problem discussed in Section 2.6. A *min-heap* is created to maintain the order of the tasks as per their *time of arrival* at the system and it is represented as HAT. The load status of the computing node M_j is represented as CL_j. Every iteration assigns the task with least arrival time to a computing node M_j in the HDCS with $CL_j = Null$.

Algorithm 3.4 FCFS

Require: T : *set of task*, M : *set of node*, ETC : *expected time to compute*, HAT : *task Queue*

Ensure: A : *Allocation List*, L : *makespan*

1: $L_j \longleftarrow 0$ *for all* node M_j
2: $A(j) \longleftarrow \phi$ *for all* node M_j
3: **repeat**
4: let t_i is the task at root of min-heap HAT
5: $allocate \longleftarrow false$
6: **repeat**
7: **for** $j = 1$ to MaxNode **do**
8: **if** $CL_j =$ Null **then**
9: Allocate task t_i to Node M_j
10: Remove task t_i from min-heap HAT
11: $A(j) \longleftarrow A(j) \cup \{t_i\}$
12: $L_j \longleftarrow L_j + t_{ij}$
13: $allocate \longleftarrow true$
14: **end if**
15: **end for**
16: **until** allocate = false
17: **until** HAT is not empty
18: $L \longleftarrow max_j L_j$

3.6.2 Randomized algorithm

A *randomized algorithm* is defined as an algorithm that is allowed to access a source of independent, unbiased random bits, and it is then allowed to use these random bits to influence its computation [125]. *Randomized algorithms* are classified into two class as *Monte Carlo algorithms* and *Las Vegas algorithms*. A Monte Carlo algorithm runs for a fixed number of steps for each input and produces an answer that is correct with a bounded probability, whereas a *Las Vegas algorithm* always produces the correct answer, but its runtime for each input is a random variable whose expectation is bounded. The

randomized algorithm used in this thesis is a Monte Carlo algorithm that runs for a fixed number of steps equal to the maximum number of tasks to be assigned.

A random allocation of task to computing nodes in an HDCS is based on random selection of the computing node. The results produced by randomized algorithms are not optimal, but are characterized by certain probability to represent the average case. Hence these are used to compare the performance of other deterministic algorithms. The details of a randomized resource allocation algorithm is shown in Algorithm 3.5. Each iteration select a task from the root of min-heap HAT and allocates it to a randomly selected computing node. The time complexity of Algorithm 3.5 is $\bigcirc(n)$ to assign n tasks to m computing node.

Algorithm 3.5 Random

Require: T : *set of task*, M : *set of node*, ETC : *expected time to compute*, HAT : *task queue*

Ensure: A : *Allocation List*, L : *makespan*

1: $L_j \longleftarrow 0$ *for all* node M_j
2: $A(j) \longleftarrow \phi$ *for all* node M_j
3: **repeat**
4: let t_i is the task at root of min-heap HAT
5: let M_j be a node selected at random
6: allocate task t_i to Node M_j
7: $A(j) \longleftarrow A(j) \cup \{t_i\}$
8: $L_j \longleftarrow L_j + t_{ij}$
9: remove task t_i from min-heap HAT
10: **until** HAT is not empty
11: $L \longleftarrow max_j L_j$

3.6.3 MINMIN algorithm

The MINMIN algorithm is a dynamic task allocation algorithm in an HDCS operate on batch mode, and is simulated through discrete event simulation [126]. Min-Min heuristics uses the ETC matrix to compute the completion time for n number of tasks. Algorithm 3.6 represents a heuristic-based algorithm for an HDCS and is named as MINMIN. This algorithm considers all the unmapped tasks during each allocation decision but maps only one task at a time.

Every allocation of a task to the computing node is followed by the update of expected completion time of all of the unallocated tasks. Let the task t_k have the minimum expected completion time on node M_l, *i.e.* $C_{kl} = min\,(C_{k1}, C_{k2}, \cdots, C_{km})$. Algorithm 3.6 allocates the task t_k to the computing node M_l as the task t_k has the minimum

Algorithm 3.6 MINMIN

Require: $T : set\ of\ task, M : set\ of\ node, ETC : expected\ time\ to\ compute$
Ensure: $A : Allocation\ List, L : makespan$

```
for all task t_i in meta-task T do
    for all machine M_j in M do
        C_ij ⟵ t_ij + L_j
    end for
end for
repeat
    for all task t_i in T do
        find the task with minimum completion time. Let t_k be the task with minimum
        completion time on node M_l
    end for
    assign task t_k to node M_l
    update load of node M_l as L_l ⟵ L_l + t_kl
    update C_il for all unallocated task
    Remove task t_k from task list T
until T is not empty
L ⟵ max_j L_j
```

expected completion time with node M_l. The *makespan* is computed after the complete allocation of all of the tasks as $L = max_j L_j$.

3.6.4 MINMAX algorithm

Algorithm 3.7 is composed of two steps. The algorithm operates on the batch of tasks and the respective ETC matrix. The algorithm computes the expected completion time for all the tasks on the HDCS. The first step is the selection of the task with minimum expected completion time in the HDCS with m nodes, Let t_k be the task with the minimum completion time on node M_l. The first step is the same as the MINMIN algorithm. The second step decides the allocation of the task to a computing node, which can be decided as follows:
If $\frac{t_{kf}}{t_{kl}} \geq C_{kl}$ then allocate the task t_k to node M_f, else assign the task t_k to node M_l.

This algorithm is different from the common max-min algorithm defined in [92, 95]. The above four algorithms mainly focus on the *makespan* for the meta-task. The *makespan*, is the total length of the scheduling, or equivalently the time when the first task starts executing, subtracted from the time when the last task completes the execution.

Algorithm 3.7 MINMAX

Require: T : *set of task*, M : *set of node*, ETC : *expected time to compute*, HTA : *taskqueue*

Ensure: A : *Allocation List*, L : *makespan*

1: **for all** task t_i in meta-task T **do**
2: **for all** machine M_j in M **do**
3: $C_{ij} \longleftarrow t_{ij} + L_j$
4: **end for**
5: **end for**
6: **for all** task t_i in T **do**
7: find the task with minimum completion time, Let t_k be the task with minimum completion time on node M_l
8: **end for**
9: **repeat**
10: **if** $\frac{t_{kf}}{t_{kl}} \geq C_{kl}$ **then**
11: assign task t_k to node M_f
12: update load of node M_f
13: update C_{if} for all i
14: **else**
15: assign task t_k to node M_l
16: update load of node M_l as $L_l \longleftarrow L_l + t_{kl}$
17: update C_{il} for all unallocated task
18: **end if**
19: Remove task t_k from task list T
20: **until** T is not empty
21: $L \longleftarrow max_j L_j$

3.7 Results and discussion

We have conducted extensive simulation with the in-house simulator. Queueing model to simulate the task arrival with the heterogeneous computing nodes. The tasks are arriving with a rate λ to the central server queue. The queue length of the central server is assumed to be infinite. We consider only 500 tasks for this experiment that uses consistent and inconsistent task models as suggested in [24]. We consider four types of heterogeneous systems. The HDCS are characterized by the service rates of the computing nodes. The four types of systems used to study the impact of heterogeneity are:

- *Type I*: All of the computing nodes are homogeneous in the system having similar architectures with an average processing rate μ.
- *Type II*: The system has two different types of computing nodes, such that $m/2$

number of computing nodes are with a service rate μ and other half of the nodes are with service rate 2μ.

- *Type III*: In the third system model half of the computing node are homogeneous and other half of the computing nodes are heterogeneous with different service rates.
- *Type IV*: The forth system model includes the computing nodes with different service rates.

We have also assumed that tasks are independent and can be processed by any computing node in distributed computing environment. For stability, it is also assumed that tasks must not be generated faster than the HDCS can process, and that the total task arrival rate at the system must not exceeds its processing rate. Therefore, the stability equations for four type of HDCS can be stated as follows:

- For *type-I* system, $\lambda \leq n\mu$.
- For *type-II* system, $\lambda \leq \Sigma_{j=1}^{m/2}\mu + \Sigma_{j=1}^{m/2}2\mu$.
- For *type-III* system, $\lambda \leq \Sigma_{j=1}^{m/2}\mu + \Sigma_{j=1}^{m/2}\mu_j$.
- For *type-IV* system, $\lambda \leq \Sigma_{j=1}^{m}\mu_j$.

The computing node heterogeneity are managed by varying the values for R_{task} and R_{mach} while generating the ETC matrix. This is realized through uniform distributions $U(1, R_{task})$ and $U(1, R_{mach})$ applied in Algorithm 3.3. Our experiments uses a fixed value for $R_{task} = 500$. The distributed systems considered for the experiment uses four different vales $10, 10^2, 10^3$ and 10^5 for R_{mach} to generate the ETC matrix representing the four different node heterogeneities. The initiation of dynamic task allocation process begins when the number of tasks waiting in the queue is grater than or equal to a batch size. We have considered three different arrival rates of tasks at the central scheduler. The three scenario are refereed in this chapter as *slow arrival*, *moderate arrival* and *fast arrival* with mean 0.1, 0.06 and 0.05 respectively. The results are obtained as an average of ten simulations. The dynamic task allocation is realized with batch mode heuristic. We have assumed that the number of tasks waiting with the central server are always greater than the batch size to facilitate discrete event simulation. As computing nodes are characterized by the processors associated with the nodes, we have assumed that computing nodes are represented by processors. In this thesis we use the term *node* and *processor* interchangeably. The first set of four experiments are conducted to determine

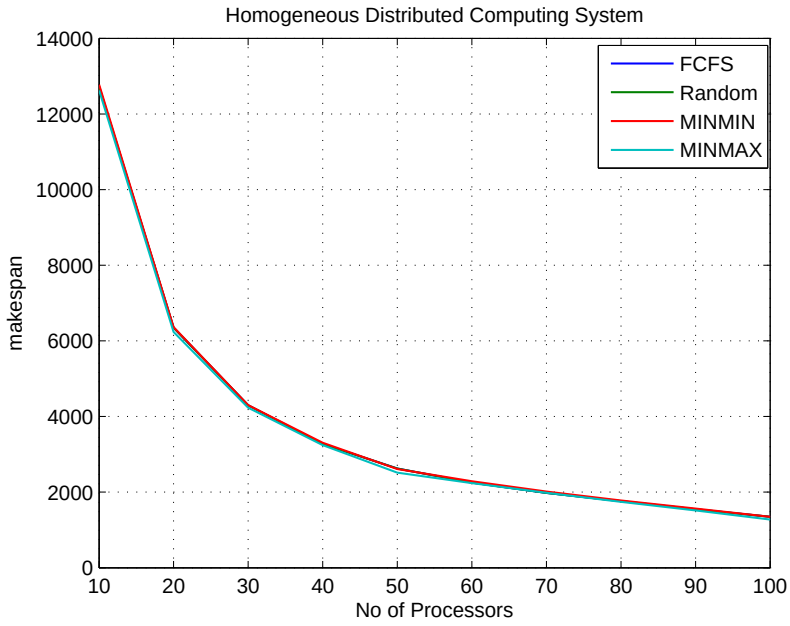

Figure 3.2: Makespan according to the number of processors in *type-I system*

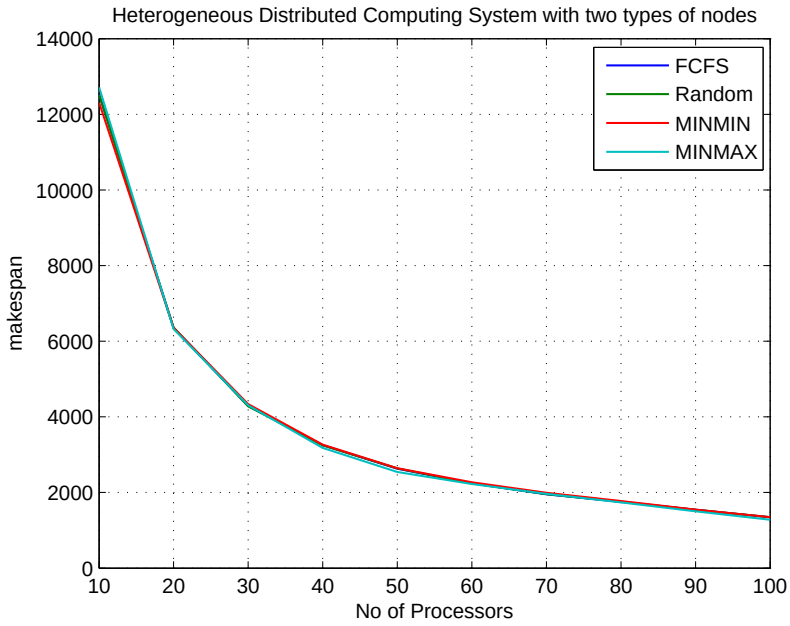

Figure 3.3: Makespan according to the number of processors in *type-II system*

the number of nodes or processors for further analysis of *greedy heuristic task allocation* algorithms in the HDCS.

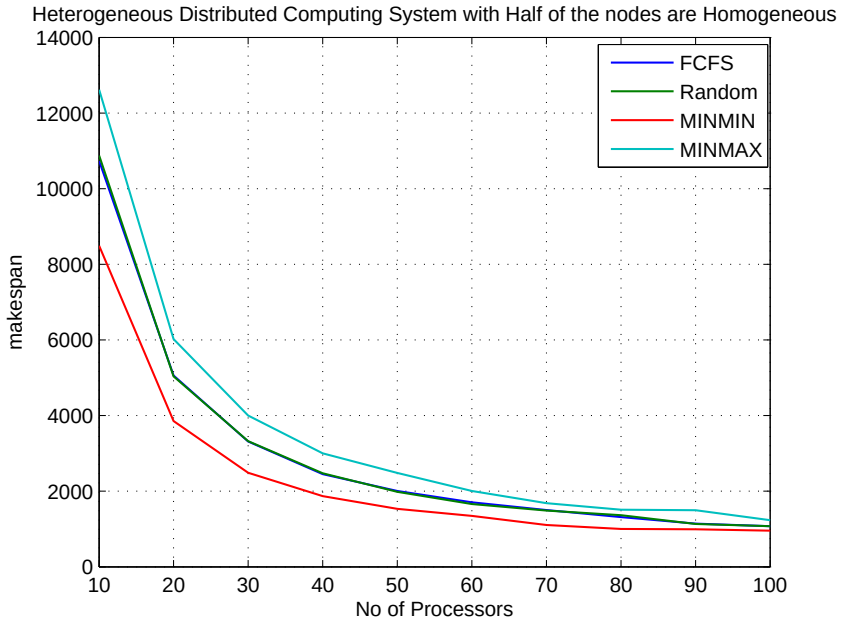

Figure 3.4: Makespan according to the number of processors in *type-III system*

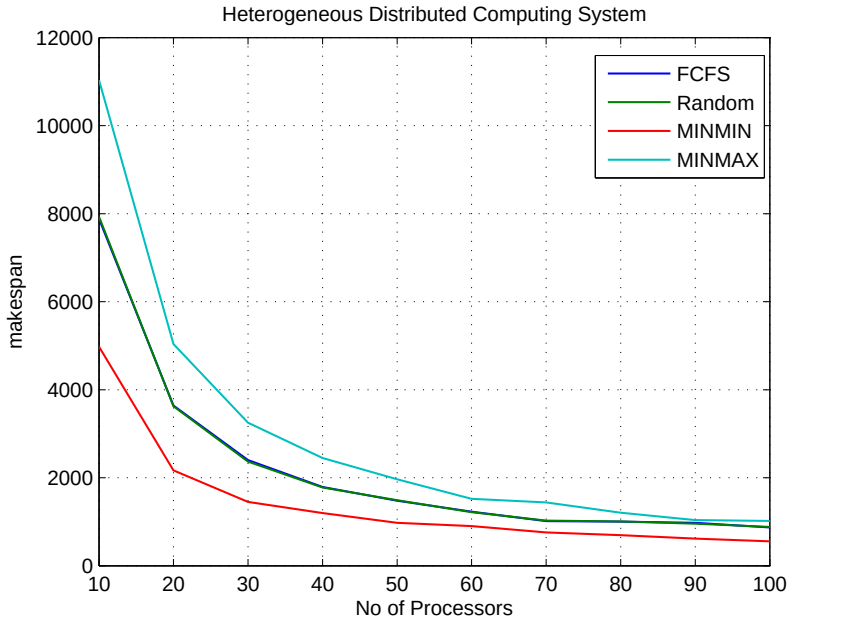

Figure 3.5: Makespan according to the number of processors in *type-IV system*

The number of node or processor for further experiments have been decided by analysing the simulation results through the graph. The central scheduler or load balancing service uses four greedy scheduling algorithms as listed in Section 3.6 for dynamic allocation of tasks in the batch mode. The simulation uses a *consistent* heterogeneous task matrix along with *arrival time* of each task and varys the number of nodes up to 100 on four different types of HDCSs with increasing heterogeneity of computing nodes. Figures 3.2, 3.3, 3.4 and 3.5 shows the comparison of the *makespan* of *FCFS*, *Random*, *MINMIN* and *MINMAX* respectively by varying computing nodes from 10 to 100. It is observed from Figure 3.5 that the *makespan* value is nearly unchanged after 60 computing nodes for the best performing MINMIN allocation algorithm; hence, 60 computing nodes are used to study the performance of resource allocation algorithms on four different HDCS environment.

3.7.1 Experiments and results with consistent ETC

This section presents the performance of different greedy heuristic resource allocation algorithms to minimize the *makespan* to meet the objective of load balancing problem as discussed in Section 2.6. A series of experiments have been conducted using discrete event simulation on four different types of distributed computing system models as mentioned in Section 3.7. For optimal load balancing, the *makespan* is minimized. Let the computing node M_1 be the fastest computing node and M_m be the slowest computing node in the HDCS. This results in a consistent ETC matrix for n number tasks on m nodes, so that $t_{ij} < t_{ik}$ for task t_i on machine M_j and M_k, with $\mu_j \geq \mu_k$. Hence for task t_i, we have $t_{i1} < t_{i2} < \ldots < t_{im}$. We have presented the performance of heuristic algorithms on four different types of HDCSs in term of machine heterogeneity. The work load parameter is the arrival rate λ of the task to the central scheduler and the expected time to compute the task on different computing nodes. The system was evaluated with slow, medium and fast loads. The arrival rate of tasks are assumed to be 10, 20 and 30 for slow, medium, and fast arrivals respectively.

3.7.1.1 Greedy heuristic algorithms on *type-I* system

Figures 3.6, 3.7 and 3.8 shows the performance of heuristic algorithms on the HDCS with all computing nodes having identical service rates. The resource allocation heuristics has no significant impact on the task arrival rate when all of the nodes in the distributed computing system are homogeneous. The plotted *makespan* value indicates in favour of

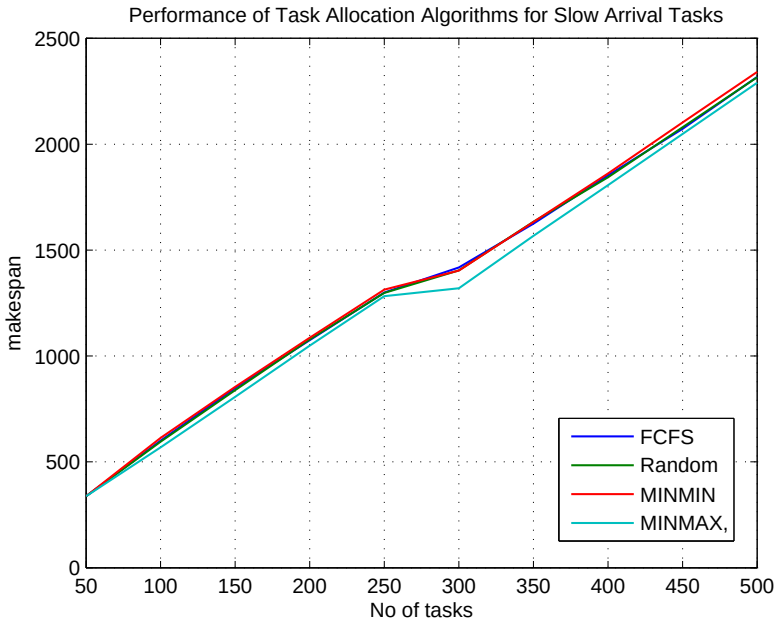

Figure 3.6: Makespan with varying number of tasks in *type-I system* for slow arrival

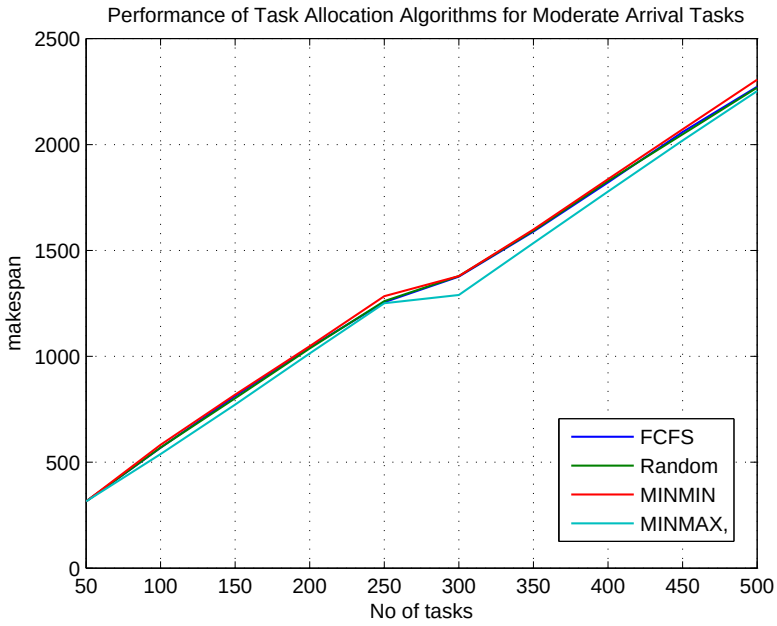

Figure 3.7: Makespan with varying number of tasks in *type-I system* for medium arrival

the MINMAX heuristic. However, the dependency of *makespan* on the number of tasks is clearly indicated by the results.

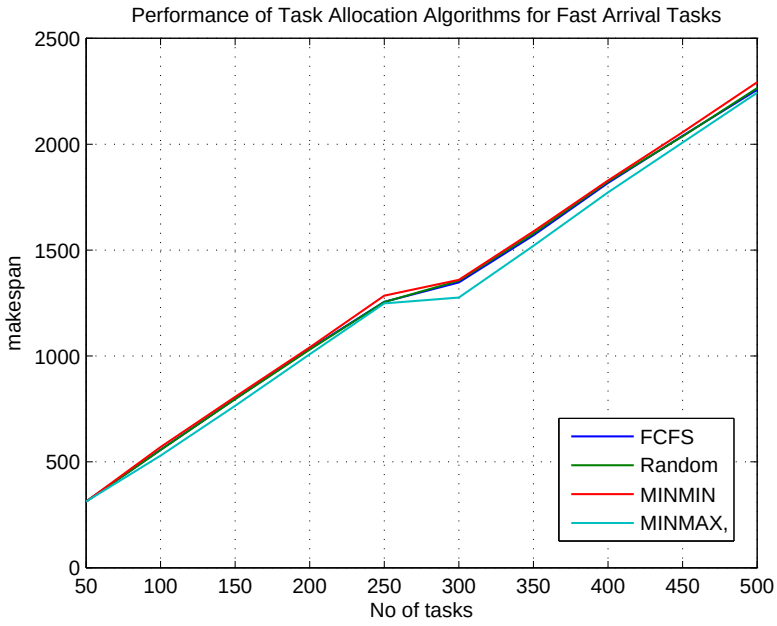

Figure 3.8: Makespan with varying number of tasks in *type-I system* for fast arrival

3.7.1.2 Greedy heuristic algorithms on *type-II* systems

These experiments are based on the model of distributed computing environments with two types of computing nodes. We have taken half of the computing nodes with a service rate μ, and the other nodes with a service rate 2μ. Figures 3.9, 3.10 and 3.11 shows the performance of heuristic algorithms on a *type-II* system. The performance of heuristic algorithms are very much similar to that observed with *type-I* systems. Plotted values of *makespan* show that the MINMAX heuristic has a better performance compared to the other schemes. It is also observed that the task arrival rate does not have a significant impact.

3.7.1.3 Greedy heuristic algorithms on *type-III* systems

The *type-III* HDCS model is for a system with equal number of homogeneous and heterogeneous computing nodes. The dependency of different load balancing heuristics on the task arrival are shown in Figures 3.12, 3.13 and3.14. Moreover, results indicate in favour of *MIMMIN*. The performance of the *MINMIN* heuristic shows improvement with a higher task arrival rate. The results obtained also indicates that *FCFS* and *Randomized* algorithms exhibit similar performance.

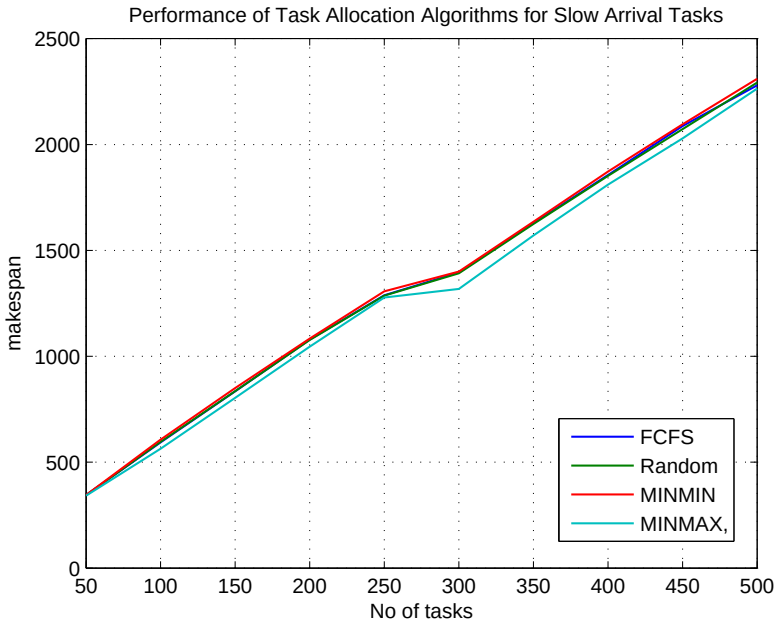

Figure 3.9: Makespan with varying number of tasks in *type-II system* for slow arrival

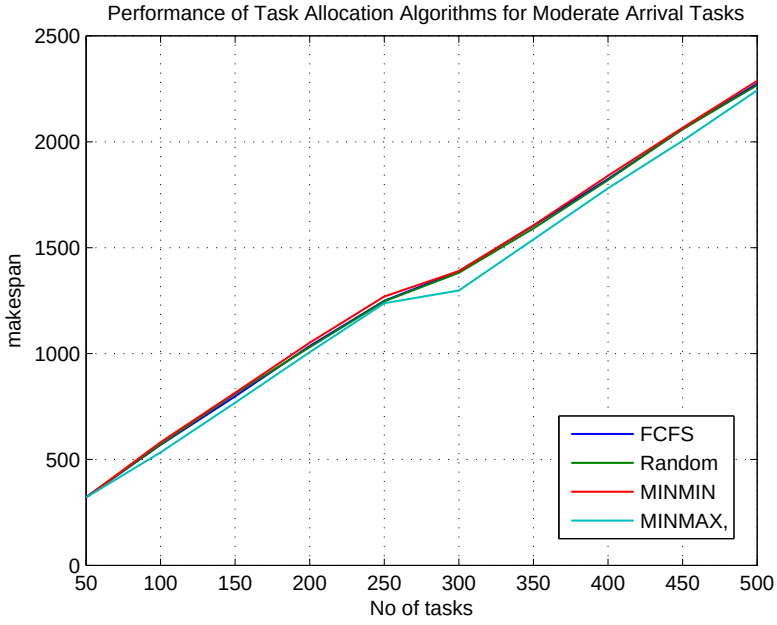

Figure 3.10: Makespan with varying number of tasks in *type-II system* for medium arrival

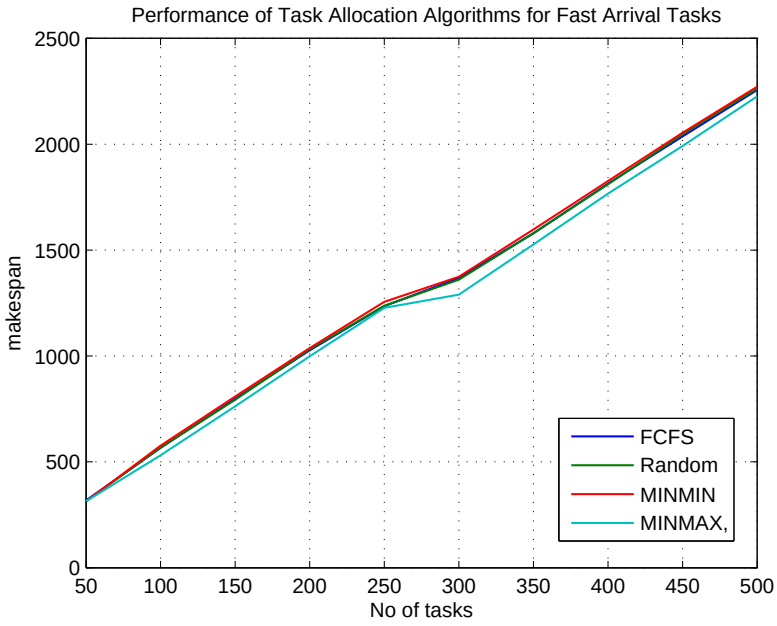

Figure 3.11: Makespan with varying number of tasks in *type-II system* for fast arrival

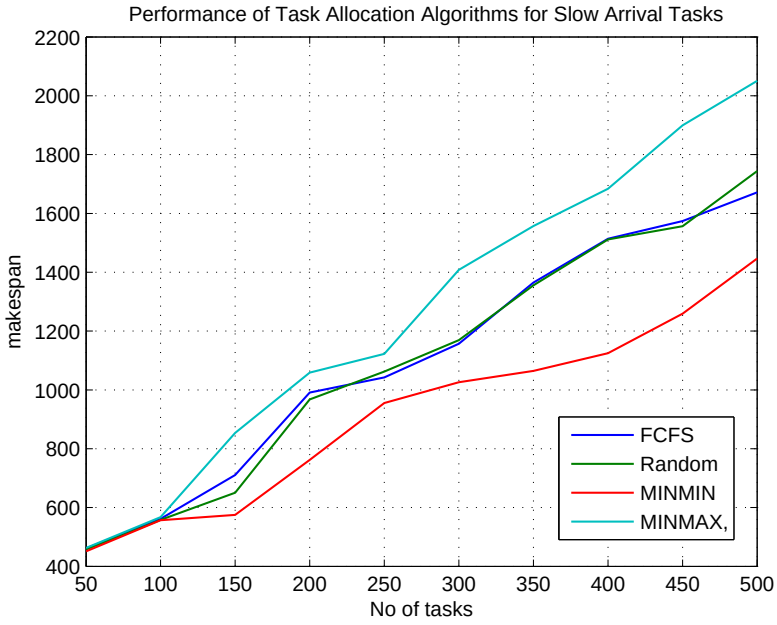

Figure 3.12: Makespan with varying number of tasks in *type-III system* for slow arrival

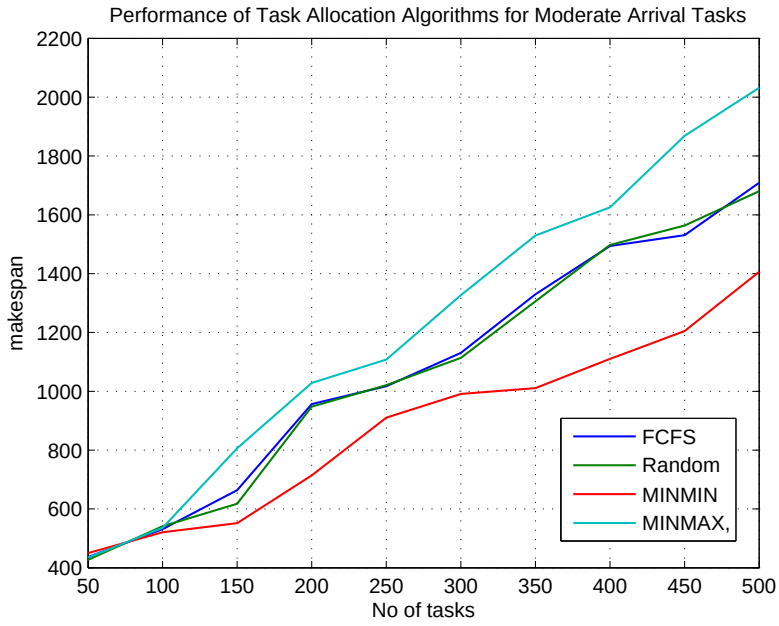

Figure 3.13: Makespan with varying number of tasks in *type-III system* for medium arrival

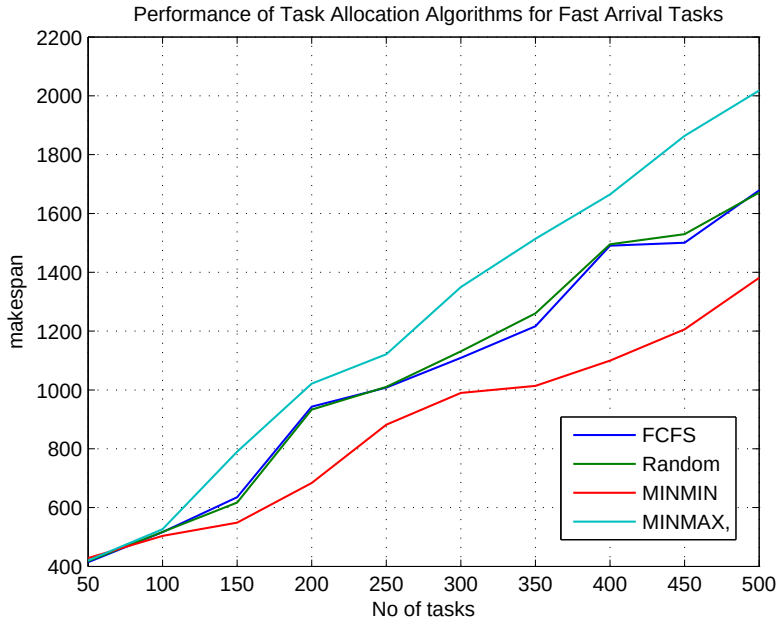

Figure 3.14: Makespan with varying number of tasks in *type-III system* for fast arrival

3.7.1.4 Greedy heuristic algorithms on *type-IV* system

This HDCS models are with heterogeneous computing nodes. Figures 3.15, 3.16 and 3.17 shows experimental results of four heuristic-based algorithms on a heterogeneous

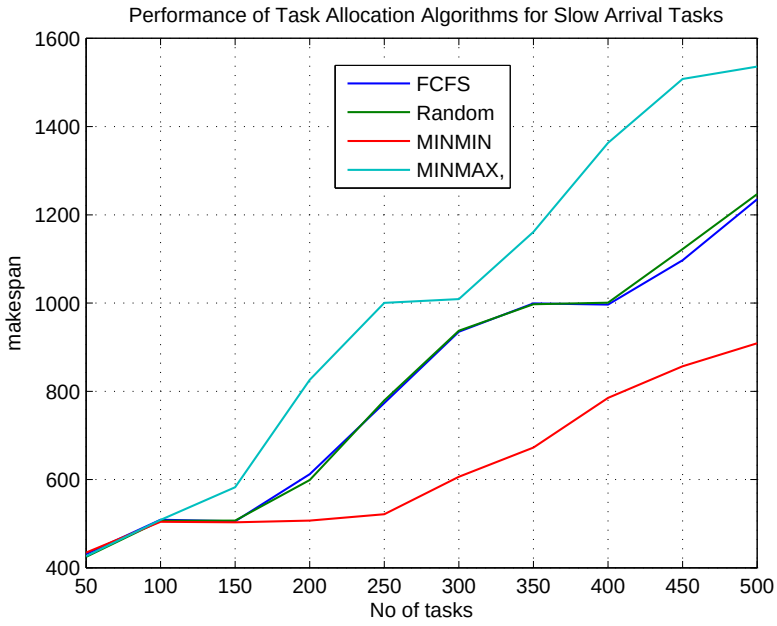

Figure 3.15: Makespan with varying number of tasks in *type-IV system* for slow arrival

system with 60 nodes. It is observed that the MINMIN algorithm significantly minimizes *makespan* over the three alternatives. Similar performance of makespan is observed for both FCFS and Randomized algorithms.

Simulation experiment were conducted with batch mode scheduler MINMIN and MINMAX separately for type-I and type-IV systems to study the impact of heterogeneity with 60 nodes An interesting observation made from Figure 3.18 is that the MINMIN algorithm shows about 56% lower *makespan* while balancing the load on 60 heterogeneous computing nodes. The simulation is conducted by varying the number of tasks. Dependency of *makespan* on the number of tasks can also be seen from Figure 3.19. It has been shown that the MINMAX algorithm on heterogeneous computing systems produces better results in comparison homogeneous computing systems with identical computing nodes with approximately 30% lower *makespan* value. MINMIN produces optimal performance with the heterogeneity of computing resources.

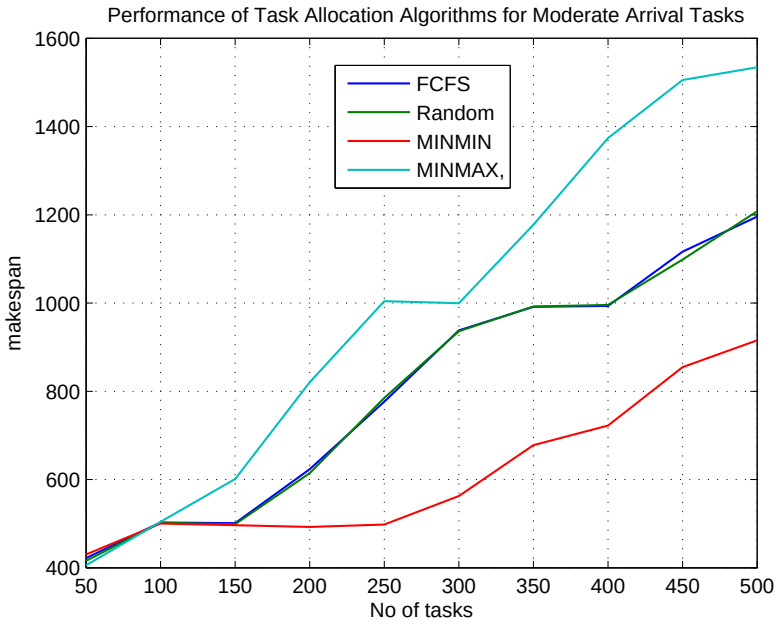

Figure 3.16: Makespan with varying number of tasks in *type-IV system* for medium arrival

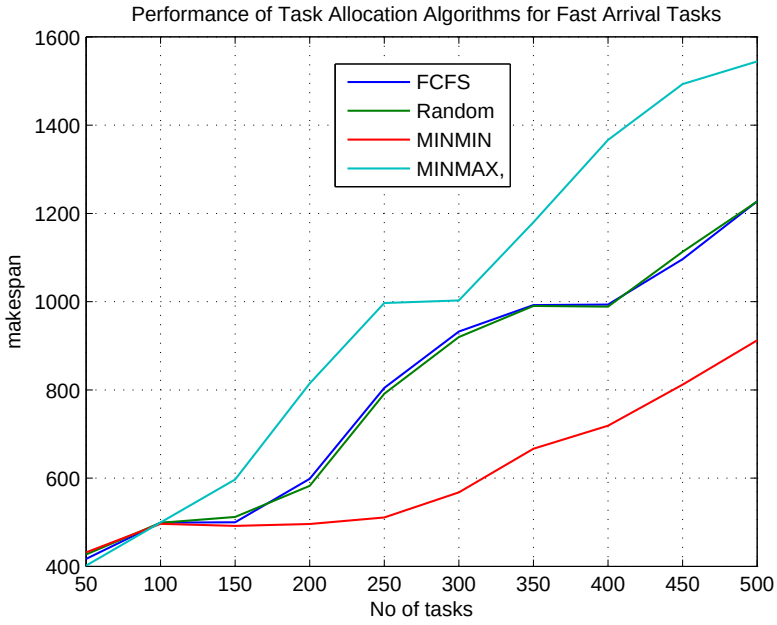

Figure 3.17: Makespan with varying number of tasks in *type-IV system* for fast arrival

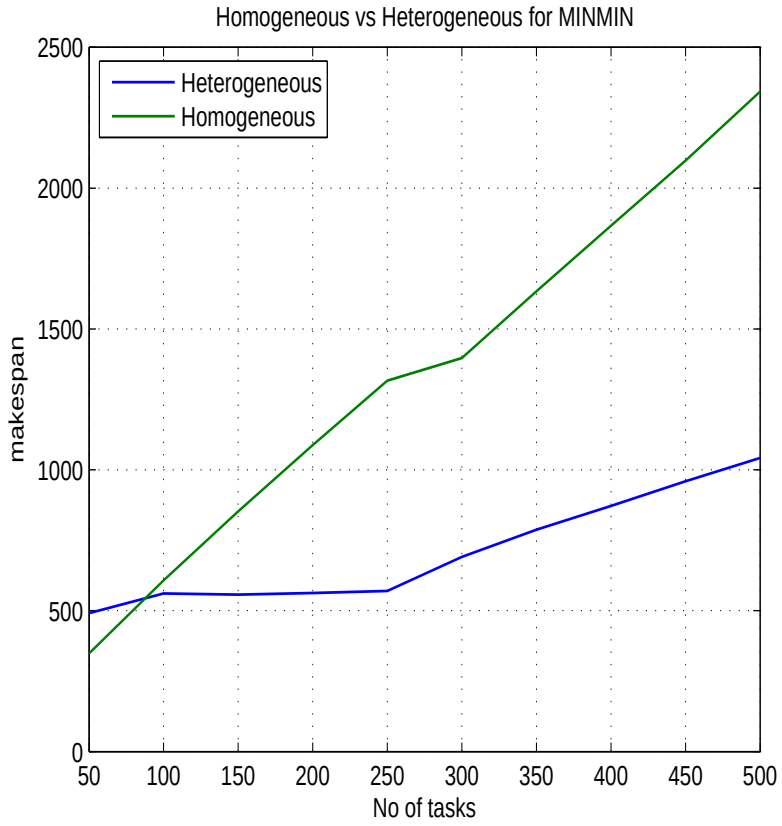

Figure 3.18: Impact of system heterogeneity using MINMIN algorithm

3.7.2 Experiments and results with an inconsistent ETC matrix

We have used an inconsistent ETC matrix to study the impact of heterogeneity in the HDCS. The simulation process is similar to that discussed in Section 3.7.1.

3.7.2.1 Greedy heuristic algorithms on *type-I* system with an inconsistent ETC matrix

Figures 3.20, 3.21, and 3.22 give a pictorial representation of the assignments made for 500 tasks on 60 computing nodes with a task arrival rate equal to 0.1. The MINMAX heuristic shows a better performance for minimizing the *makespan* in systems with homogeneous nodes.

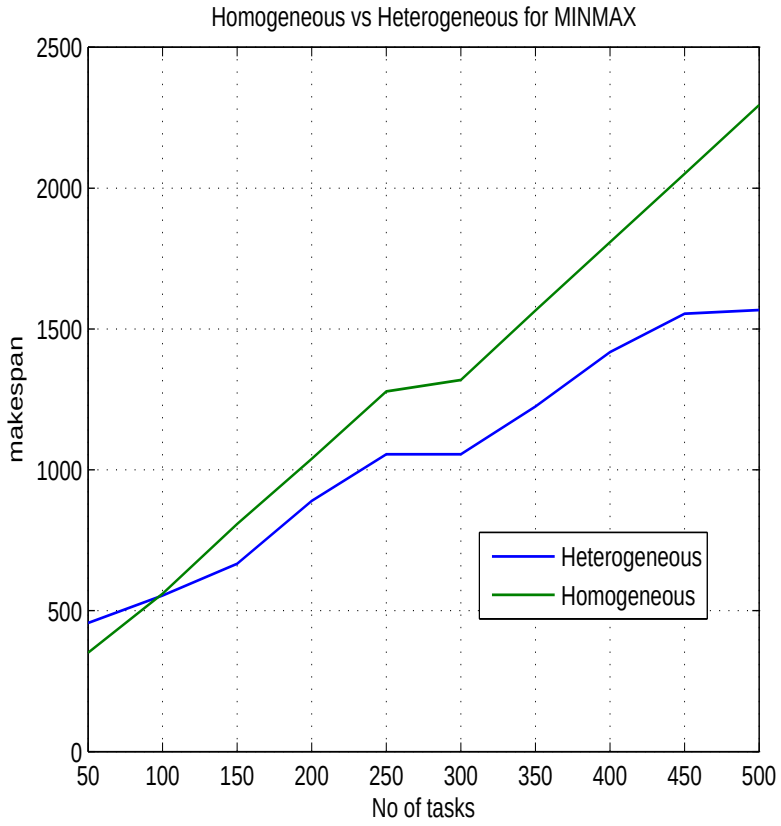

Figure 3.19: Impact of system heterogeneity using MINMAX algorithm

3.7.2.2 Greedy heuristic algorithms on *type-II* system with an inconsistent ETC matrix

Figures 3.23, 3.24 and 3.25 show the performance of heuristic algorithms on *type-II* systems using an inconsistent matrix. Plotted *makespan* values show the better performance for diffident arrival rates. A clear difference in performance of MINMIN and MINMAX can be observed. Also, it observed that task arrival rate has no significant impact on task allocation ability of the algorithms.

3.7.2.3 Greedy heuristic algorithms on *type-III* system with an inconsistent ETC matrix

The impact of an inconsistent ETC matrix are shown in Figures 3.26, 3.27 and 3.28. The graphs shows the impact of heterogeneity on resource allocation algorithms. The MINMIN heuristic performs better with more number of tasks in the system at a particular

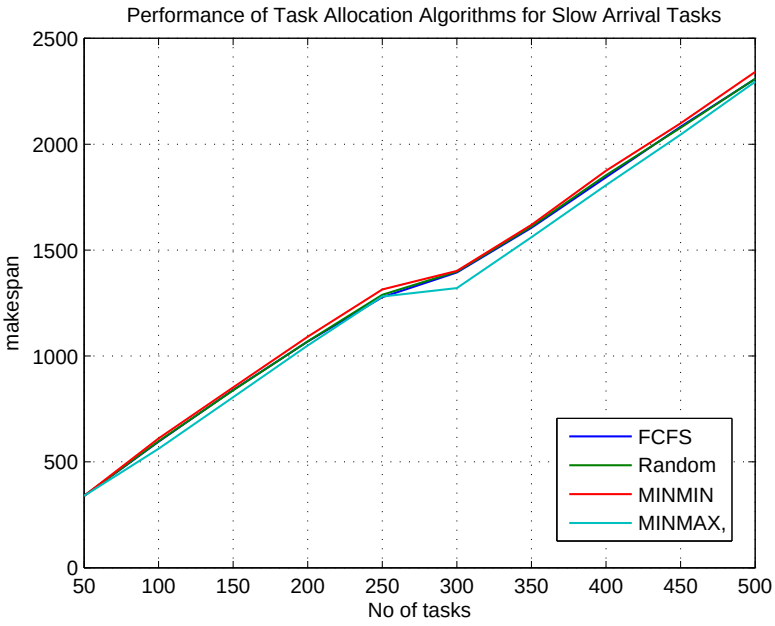

Figure 3.20: Makespan with varying number of tasks in *type-I system* with slow arrival of inconsistent task

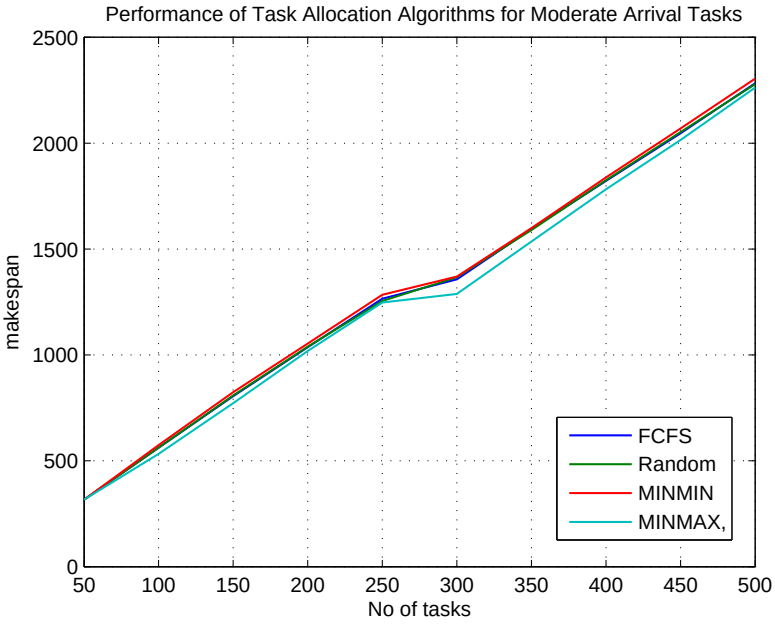

Figure 3.21: Makespan with varying number of tasks in *type-I system* with medium arrival of inconsistent task

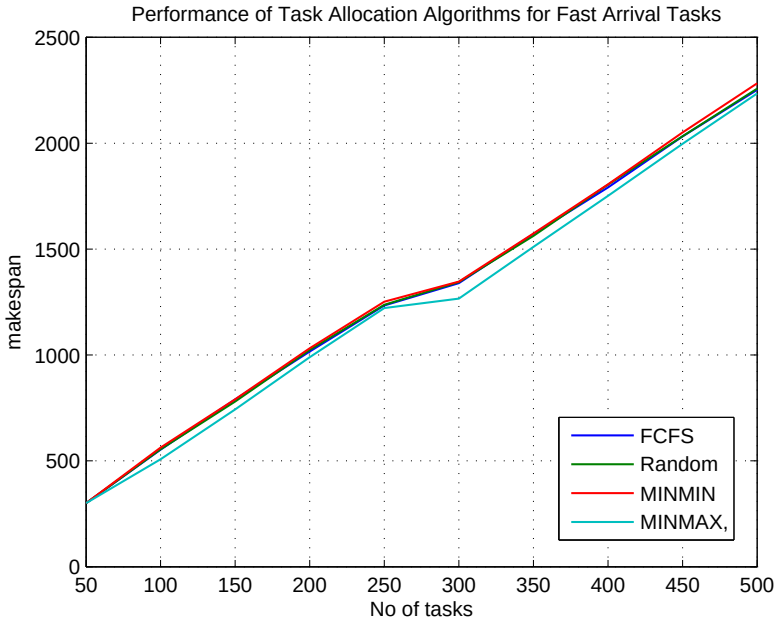

Figure 3.22: Makespan with varying number of tasks in *type-I system* with fast arrival of inconsistent task

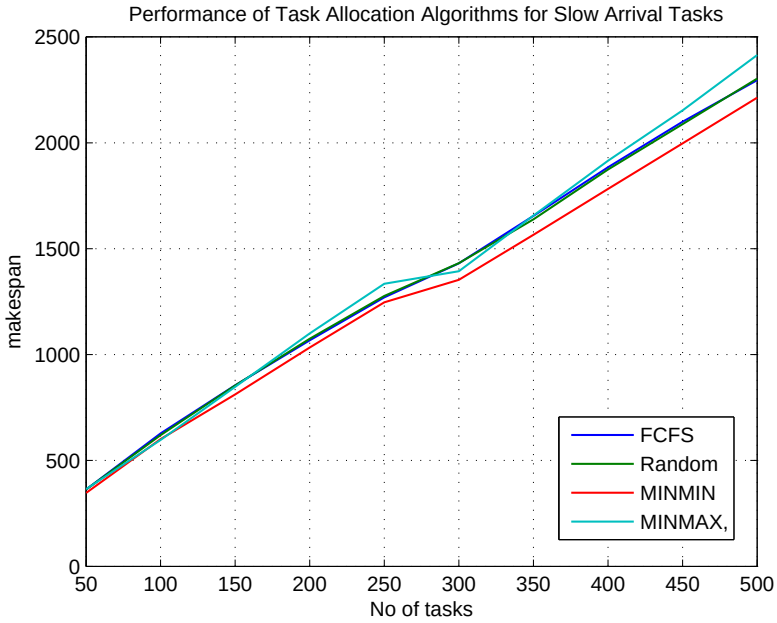

Figure 3.23: Makespan with varying number of tasks in *type-II system* with slow arrival of inconsistent task

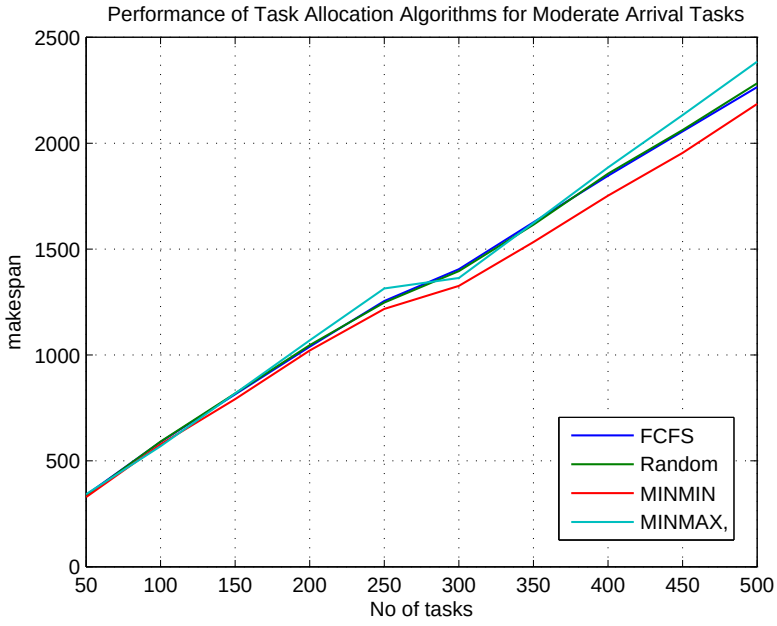

Figure 3.24: Makespan with varying number of tasks in *type-II system* with medium arrival of inconsistent task

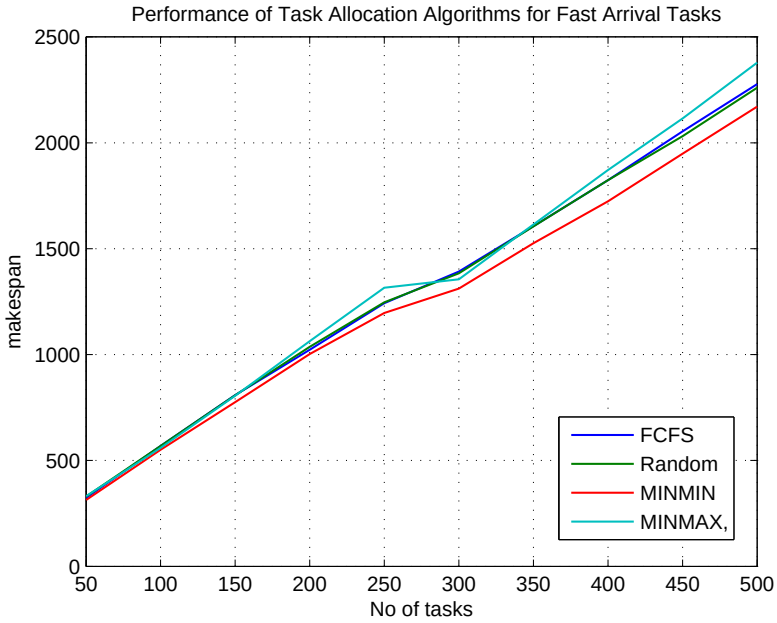

Figure 3.25: Makespan with varying number of tasks in *type-II system* with fast arrival of inconsistent task

time instant. The results obtained also indicate that *FCFS* and *Randomized* algorithms exhibit similar performance on allocating the tasks for an inconsistent task model.

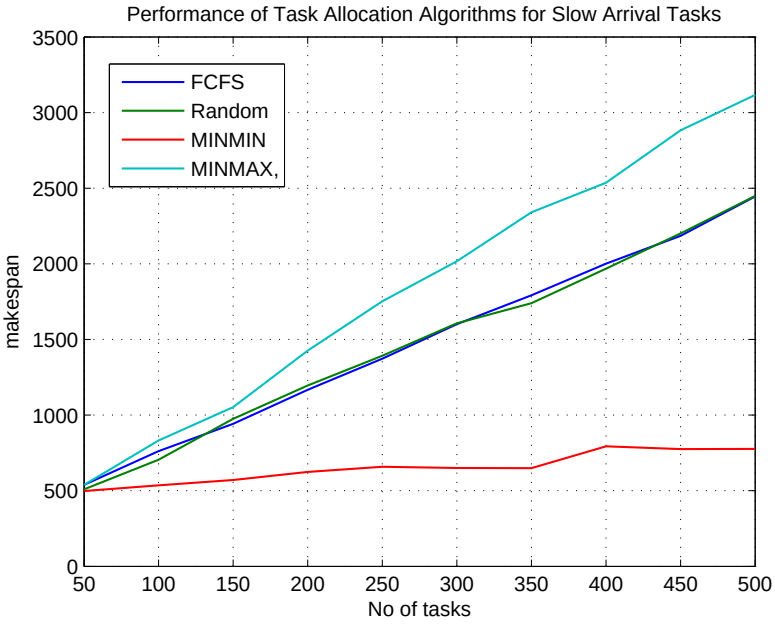

Figure 3.26: Makespan with varying number of tasks in *type-III system* with slow arrival of inconsistent task

3.7.2.4 Greedy heuristic algorithms on *type-IV* system with inconsistent ETC matrix

Simulation results on the HDCS with heterogeneous computing nodes are shown in Figures 3.29, 3.30 and 3.31. The simulation results with 60 nodes clearly indicate the better performance of the MINMIN algorithm. Moreover, in three alternatives, both FCFS and Randomized algorithms exhibit similar performance in terms of *makespan*. The highest *makespan*, average *makespan* and minimum *makespan* value are indicated against varying task size.

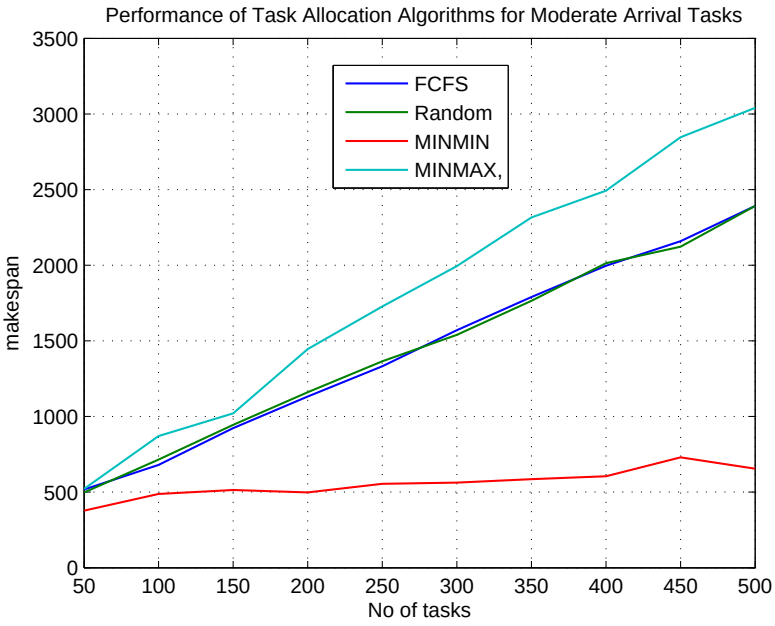

Figure 3.27: Makespan with varying number of tasks in *type-III system* with medium arrival of inconsistent task

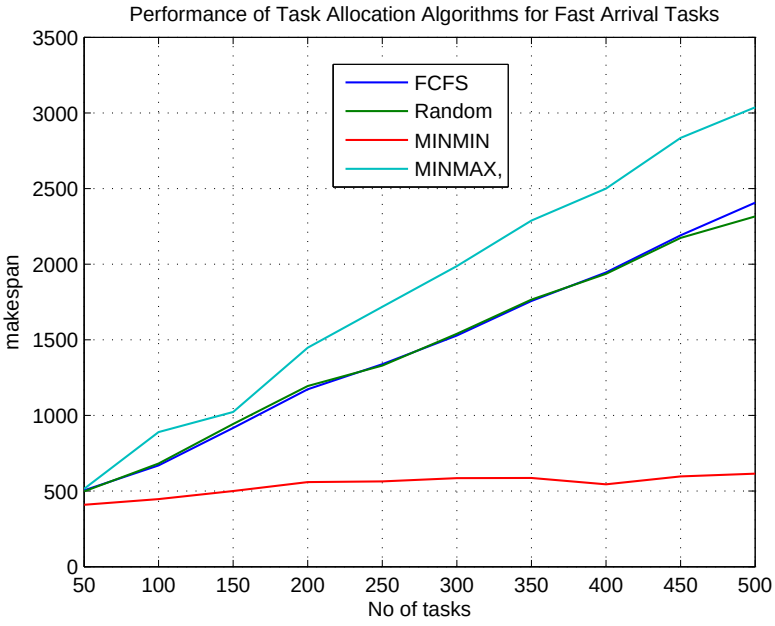

Figure 3.28: Makespan with varying number of tasks in *type-III system* with fast arrival of inconsistent task

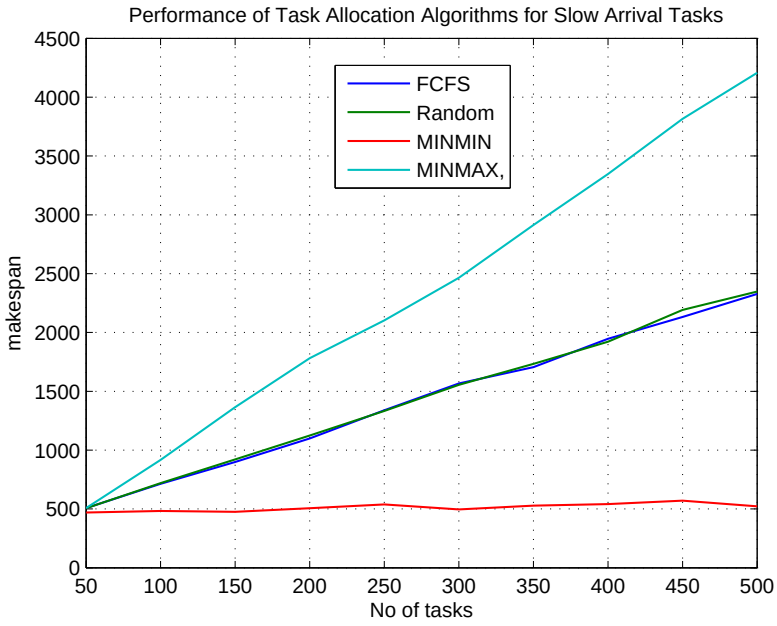

Figure 3.29: Makespan with varying number of tasks in *type-IV system* with slow arrival of inconsistent task

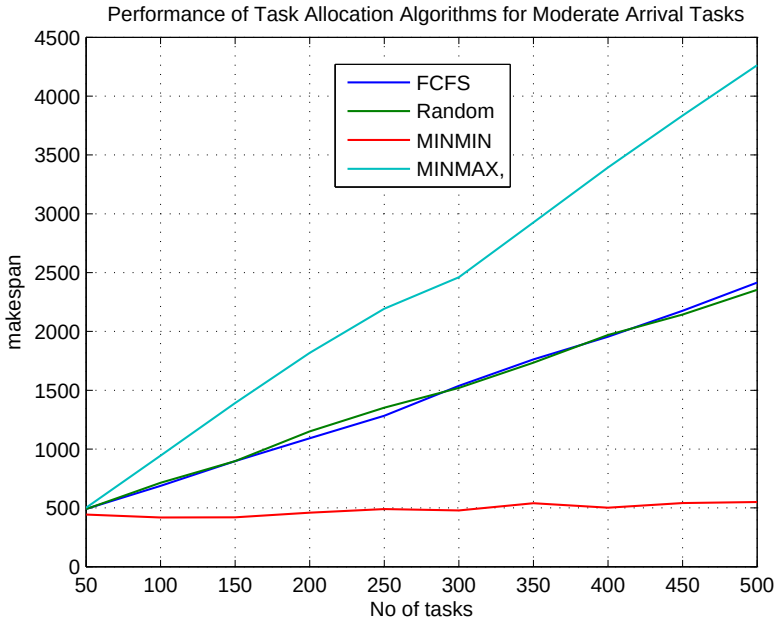

Figure 3.30: Makespan with varying number of tasks in *type-IV system* with moderate arrival of inconsistent task

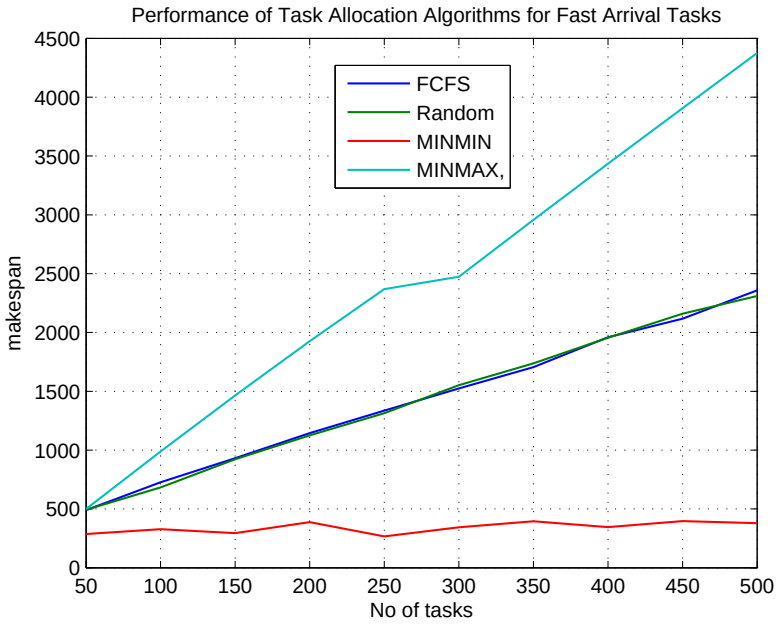

Figure 3.31: Makespan with varying number of tasks in *type-IV system* with first arrival of inconsistent task

3.8 Conclusion

A number of experiments are conducted to examine the performance of greedy resource allocation algorithms against *makespan* to study the task and node heterogeneity in HDCS by considering three different arrival rates for the tasks. Greedy algorithm paradigm used for load balancing performs better as it uses simple heuristics for task allocation. An average case analysis is presented in this chapter through simulation. The experiments conducted assuming with inconsistent and consistent ETC matrix both lead to conclusive observations on task and machine heterogeneity. Analytical models and simulation studies demonstrates the performance of the load balancing algorithms, and these results have been confirmed in different distributed systems models. In the *batch mode*, the central scheduler considers a meta-task for matching and scheduling at each mapping event. We have been able to establish that the *batch mode* mapping heuristics make better decisions, because the heuristics have the resource requirement information for the meta-task, and know the actual execution time of a larger number of tasks (with higher arrival rate). It is observed that the performance of the greedy scheduling algorithm is also affected by the rate of heterogeneity of the task and computing nodes as well as consistency of the tasks.

Chapter 4

Stochastic Iterative Algorithms for Load Balancing

Genetic algorithms and simulated annealing are search techniques that can be applied to produce sub-optimal solutions for various NP-complete problems. These two algorithms are used to solve the load balancing problem, which is the optimal dynamic allocation of tasks on distributed computing system. A new codification scheme suitable to simulated annealing *and* genetic algorithm *has been introduced to design dynamic load balancing algorithms for an HDCS. The resource allocation algorithms use sliding window techniques to select the tasks to be allocated to computing nodes in each iteration. Simulated annealing and genetic algorithm frameworks for dynamic load balancing are explained along with implementation details. The task model used is based on Expected Time to Compute matrix. The effect of genetic algorithm based dynamic load balancing scheme has been compared with first-fit, randomized heuristic and simulated annealing through simulation.*

4.1 Introduction

Distributed heterogeneous computing is being widely applied to a variety of large sized computational problems. This computational environments consist of multiple heterogeneous computing modules, which interact with each other to solve the problem. In an HDCS, processing loads arrive from many users at random time instants in the form of task. A proper scheduling policy attempts to assign these tasks to available computing nodes so as to complete the execution of all the tasks in the shortest possible time. Genetic Algorithms (GAs) and Simulated Annealing (SA) are used as alternative approaches

to perform exhaustive search in exponentially large solution spaces and are found to be efficient for various intractable problems. Dynamic load balancing is a classical optimization problem as defined in Equation 2.3 and aims to minimize the *makespan*. Algorithm paradigms, like branch-and-bound techniques and dynamic programming are quite effective but their time-complexity is often too high and unacceptable for NP-complete problems [127]. Very often greedy algorithms are successfully used for NP-complete problems, but the solution is sub-optimal and of low quality. Heuristic algorithms are used to overcome these disadvantages. GAs are among such techniques; they are stochastic algorithms whose search methods model some natural phenomena [128]. Jong and Spears [129] have demonstrated the application of Genetic Algorithm (GA) to solve NP-complete problems. Simulated annealing is a heuristic method that has been applied to obtain good solutions for a defined objective function through *local search* [107].

Dynamic load balancing algorithms operates in batch mode, by selecting a subset of the unscheduled task, called a batch. At each iteration, a batch of task is allocated among the computing nodes of the HDCS. The batch size is fixed and is to be decided by considering task and machine heterogeneity [106]. We have used an inconsistent ETC matrix to study the performance of schedulers based on GA and SA with the objective of minimizing the *makespan*. A genetic algorithm performs a multi-directional search by maintaining a population of potential solutions and an objective(fitness) function that plays the role of an environment [103, 128]. Dynamic load-balancing mechanisms developed using GA and SA have been compared with the *first-fit*(FF), and the randomized heuristic algorithm.

It is assumed that a computing node is a machine with a single processor or processing element or computing element. Hence, the performance of dynamic resource allocation algorithms in this chapter have been studied using *processor utilization* as the performance metric, which also indicates the performance of the computing node. To improve the utilization of the processors, the tasks are distributed among the processors in such a way that the computational load is spread evenly among the processors. We have used a centralized load balancing algorithm framework as it imposes fewer overheads on the system than the decentralized algorithm. This chapter demonstrate the use of the common coding scheme and iterative algorithmic structure with GA and SA for allocating the tasks among the computing nodes to optimize *processor utilization* and *completion time*.

4.2 Related work

The complexity of dynamic load balancing increases with the size of the HDCS and becomes difficult to solve effectively. The exponential solution space of the load balancing problem can be searched using heuristic techniques based on GA and SA to obtained sub-optimal solutions in acceptable time [6, 17, 92, 95]. These artificial intelligence techniques have been used by researchers and proven to be effective in solving many optimization problems. A review of theoretical foundation of genetic algorithms along with canonical GA and experimental forms of genetic algorithms are presented in the tutorial by Whitley [130]. GAs have been used by various researchers to obtain sub-optimal solutions to the load balancing problem in distributed systems [6, 17, 22, 57, 131, 132, 133, 134]. The SA approach, proposed by Kirkpatrick et al. [34, 101], has been used as a popular heuristic to solve several optimization problems to obtain sub-optimal solutions. SA is a heuristic method that performs local search in an exponential solution space to obtain good solution for discrete optimization problems. The search process is analogous to the annealing process of metals, that stabilizes to a low energy configuration when cooled with an appropriate cooling schedule [107, 135]. Abraham *et al.* [136] demonstrated the application of SA and the hybrid of GA and SA for job scheduling on large scale distributed systems. A load balancing scenario based on genetic algorithm has been presented by Shan and Zhou [137].

Kim and Kim [138] presented methods to solve scheduling problem in a manufacturing system by applying simulated annealing and genetic algorithms with the finite loading method. A comparative analysis of GAs, SA and hill-climbing algorithms to solve the dynamic mapping problem was addressed in [139]. A hybrid load balancing strategy using the first-come-first-served and the GA has been designed by Li *et. al* [60]. This algorithm uses the centralized scheduler model in a grid computing environment to schedule sequential tasks to achieve minimum execution time, maximum node utilization and load balancing across all the nodes. Researchers have examined eleven different heuristics, namely opportunistic load balancing, minimum execution time, minimum completion time, minmin, maxmin, duplex, genetic algorithm, simulated annealing, genetic simulated annealing, tabu search, and A* on mixed-machine heterogeneous computing environments to minimize the total execution time of the metatask [92, 95]. Rahmani and Rezvani [140] presented a GA for static scheduling, which is again improved by SA to obtain an improved solution.

4.2.1 Genetic algorithms in load balancing

Zomaya and Teh [6] proposed a dynamic load balancing framework using GA. The proposed GA load balancer uses a centralized approach to handle the load balancing decisions. Effectiveness of a central server in load balancing has been demonstrated for homogeneous distributed computing systems. A *batch-mode* genetic scheduler has been used by Page and Naughton [106, 141] and the performance has been compared with the *immediate-mode* schedulers on an HDCS. Li *et al.* [60] presented a GA based hybrid load balancing strategy considering sequential tasks for grid computing. Aggarwal *et al.* [142] have designed and tested a GA based scheduler to schedule multiple jobs with quality-of-service constraints for tasks represented as a directed acyclic graph (DAG). Genetic algorithm based scheduler for grid computing systems can also be found in [143]. Ahmad *et al.* [144] introduced a technique based on the problem-space GA for static task assignment in heterogeneous distributed systems. An evolution-based dynamic scheduling algorithm that utilizes the GA is proposed by Yu *et al.* [145] to schedule heterogeneous tasks to appropriate computing nodes in a grid computing environment. Dynamic load balancing using GA can be found in [133, 134, 146] for the tasks represented as task interaction graph. Greene [131] has designed a GA scheduling routine that produces low cost, well balanced schedules for the incoming tasks.

The use of GA and Tabu search are used to solve load balancing problem in distributed computing infrastructures called computational grids are presented in [17]. The use of GA for static mapping of tasks on heterogeneous computing environments with *deadlines*, *priorities*, and *multiple versions with subtasks* has been discussed by Braun *et al.* [111]. Greene [131] presented a dynamic load balancing genetic algorithm with three variations, *i.e.*, number of processors, number of tasks to be scheduled, and distribution duration of tasks, to minimize the *makespan* on multiple machines. *Hybrid crossover* and *incremental mutation* operations are implemented with the simple GA framework to obtain near optimal solutions. Tripathi *et al.* [22] presented GA based task allocation methods for multiple disjoint tasks in distributed computing systems. Maximizing the reliability of a distributed computing system with GA-based task allocation using GA, with the task represented as task graph, was discussed by Vidyarthi and Tripathi [134]. Lee and Hwang [147] have proposed a GA-based method for improved sender-initiated dynamic load balancing in distributed systems. GA-based generalized dimension exchange method has been discussed by Cheong and Ramachandran [148]. Page *et al.* [132] have presented a multi-heuristic evolutionary dynamic task allocation algorithm to map tasks to processors in a heterogeneous distributed system. Yu and Chen [145] have proposed an algorithm that uses the GA as a search technique for efficient scheduling in a grid computing

environment considering the heterogeneity of computing nodes.

4.2.2 Simulated annealing algorithm in load balancing

Simulated annealing has been used to solve unconstrained and bound constrained optimization problems. SA proposed by Kirkpatrick *et al.*[101] has been used as a popular heuristic to solve optimization problems. Theys *et al.* [92] has used the SA approach to solve the static load balancing problem in an HDCS. A comparative study of the three algorithms Hill-climbing, SA and GA for static placement of communicating process on the processors of a distributed memory parallel machine has been presented by Talbi and Muntean [139]. A heuristic algorithm based on SA is discussed in [149], which guarantees good load balancing in a grid environment. A classification of iterative dynamic load balancing techniques can also be found in [149]. *Makespan* minimization of scheduling problem on identical parallel machines using SA has been presented by Lee *et al.* [34]. Grid Computing is one of heterogeneous distributed computing systems, in which several entities are geographically dispersed. Fidanova [107] used SA to obtain near optimal solutions for scheduling problem in a large grid. Suman and Kumar [150] presented a survey of simulated annealing based optimization algorithms to solve single and multiobjective optimization problems. Distributed SA algorithms for job scheduling in distributed systems has been presented by Krishna *et al.* [151]. Attiya *at el.* [45] proposed a simulated annealing approach to maximize the reliability of the distributed system. A heuristic framework using SA for solving resource allocation and scheduling problem with precedence constraints is presented by Zhang *et al.*[152].

Several researchers used SA and GA for load balancing in a distributed computing system; however, majority of the work have no specific representation for simulated annealing algorithms for load balancing. It is also observed that dynamic task allocation algorithms are used to schedule the tasks in small batches. This chapter presents detail schemes suitable for designing dynamic load balancing algorithms using the GA and SA. The resource allocation decisions for n tasks on m computing nodes are realized using batch mode heuristics, with the ETC matrix representing task and machine heterogeneity.

4.3 System model

HDCS environments are well suited to meet the computational demands of large, diverse groups of tasks. We have considered the centralized HDCS model with m computing nodes under the supervision of a serial scheduler or central scheduler. The computing power of a node is with the processor associated with the node. For simplicity it has been

assumed that each computing node has a single processor. Without loss of generality, we have used the term *node* and *processor* interchangeably. The tasks on arrival are placed in the queue at the central scheduler. On each invocation of the scheduler a batch of tasks are selected and allocated to the computing nodes. In this chapter a batch of tasks is selected with a sliding window. The maximum number of task that can be selected once represents the size of the sliding window.

We have assumed that all computational tasks are capable of being executed on any of the computing nodes of the HDCS. A single computing node acts as a central scheduler or resource manager of the HDCS and collects the global load information of other computing nodes. Resource management sub systems of the HDCS are designated to schedule the execution of the tasks dynamically in a batch mode, as they arrives for the service. The system and task models are the same as discussed in Section 3.4. The HDCS is modelled as an $M/M/m$ queuing system. In particular, tasks arrive randomly to the central scheduler following a Poisson distribution with expected time of the task following a uniform distribution [91, 153]. The queue of unscheduled task with central scheduler can accommodate a large number of tasks and if all are to be assigned to the computing nodes at once, the scheduler could take a long time to find the efficient schedule. To speed-up the scheduler, and reduce the chance of a processor becoming idle, tasks are selected in a batch that matches with the window size [106]. This also refers to the current load of the computing nodes before the allocation of next batch of task.

The centralized load balancing algorithms requires the global information on computing nodes at a single location and the load balancing policy is initiated from the central location by the central scheduler. We have adopted the task model introduced by Ali *et. al* [24] to analyse the performance of stochastic iterative task allocation algorithms. In particular, the task pool is represented by an ETC matrix model as discussed in Section 2.5. The tasks are assumed to be CPU bound hence the communication overhead between central scheduler and computing nodes are negligible in comparison with the expected computation time [60]. The tasks arrive at the central scheduler following a Poisson distribution. It is also assumed that tasks are mutually independent and can be executed in any computing node. Each computing node is modeled as an $M/M/1$ non pre-emptive queue so that an executing task can not be interrupted or migrated to another computing node. Each computing node can execute one task at a time. The communication between the central scheduler and the nodes are assumed to be through message passing without any communication overhead.

The random generation of the ETC matrix are supported by researchers, as determining a representative set of HDCS task benchmarks remains a challenge for the research

Table 4.1: ETC matrix for 10 tasks on five node

Node Task	M1	M2	M3	M4	M5
t1	22	21	6	16	15
t2	7	46	5	28	45
t3	64	83	45	23	58
t4	53	56	26	42	53
t5	11	12	14	7	8
t6	33	31	46	25	23
t7	24	11	17	14	25
t8	20	17	23	4	3
t9	13	28	14	7	34
t10	2	5	7	7	6

community [92]. We have generated an inconsistent ETC matrix for 10 tasks on 5 machines with the expected computation time uniformly distributed in the interval $(1, 100)$ as shown in Table 4.1. This ETC matrix is used to explain various operations used for GA and SA.

The dynamic resource allocation routines assumes the load of the individual node L_j to be initialized to zero for all of the computing nodes in the HDCS. In the context of dynamic load allocation, the stochastic iterative algorithms operates in a batch-mode. After a batch of tasks is allocated to the computing nodes, the load of the computing nodes are updated. The expected completion time of all of the unallocated tasks are to be computed with reference to the ETC matrix as $C_{ij} \longleftarrow L_j + t_{ij}$ after each batch of allocation. As the current load of the nodes are reflected in expected completion time of each unallocated task, this dynamic resource allocation.

4.4 Encoding mechanism

The genetic algorithms and the simulated annealing algorithm requires a suitable representation and evaluation mechanism for finding a solution. At each of the iterations, scheduling of tasks to the different processors, are made in such away that the loads of the computing nodes are balanced. A task schedule (TS) is the linear representation of nodes on which the tasks are to be executed in order. We have used the structure as shown in Figure 4.1 to represent the task schedule $TS = (ts_1, ts_2, \ldots, ts_{WinSize})$, where $WinSize$ is the fixed length of the execution window. In TS, the integer value assigned to individual element of the array indicates the computing node number in the HDCS. At each of the steps, the number of tasks to be allocated to the computing node is determined by using

either the GA or the SA dynamic scheduling routine. At the time of allocation, there may be a large number of tasks are with the central scheduler. The sliding window technique is used to select a batch of tasks that fit to the window for subsequent allocation. After allocating each batch tasks, the expected completion time for all of the unallocated tasks are calculated. This represents the dynamic allocation process. Figure 4.1 represents 10 tasks and their respective allocation to five computing nodes $M_1, M_2, \ldots, M_5$. Figure 4.2 shows the structure of an individual, that indicates the computing node as a *gene* in the GA terminology. We have assumed that the current workload is represented as dedicated tasks for each of the nodes, so that the calculation of *makespan* is carried out from the time the sliding window is selected. Figure 4.2 represents ten tasks and their respective

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
M_5	**M_3**	**M_5**	**M_3**	**M_2**	**M_2**	**M_4**	**M_4**	**M_1**	**M_1**

Figure 4.1: Allocation of 10 task to 5 node

5	**3**	**5**	**3**	**2**	**2**	**4**	**4**	**1**	**1**

Figure 4.2: Individual

allocation to five computing nodes M_1, M_2, M_3, M_4, M_5. Allocation list for the ten tasks is created by including the node number to which the respective tasks are allocated. The allocation list represents the structure of a chromosome or individual, and a gene is represented by the computing node number. The chromosome structure, shown in Figure 4.1, is used to design the GA-based load balancer for task allocation.

Table 4.2: Makespan of the system with 5 node

Node	A(i)		L_i	Average utilisation
1	t(9,1)=13	t(10,1)= 02	15	0.2054
2	t(5,2)=12	t(6,2)= 31	43	0.5890
3	t(2,3)=05	t(4,3)= 26	31	0.4246
4	t(7,4)=14	t(8,4)= 04	28	0.3835
5	t(1,5)=15	t(3,5)= 58	**73**	1.0000

Table 4.3: Makespan of the system with 5 node with initial load

Node	Initial Load	A(i)		L_i	Average utilisation
1	09	t(9,1) = 13	t(10,1) = 02	24	0.3076
2	11	t(5,2) = 12	t(6,2) = 31	54	0.6923
3	07	t(2,3) = 05	t(4,3) = 26	38	0.4871
4	15	t(7,4) = 14	t(8,4) = 04	43	0.5512
5	05	t(1,5) = 15	t(3,5) = 78	**78**	1.0000

The average utilization for a computing node M_j is calculated as :

$$Average\ utilization\ of\ node\ M_j\ = \frac{makespan}{L_j} \tag{4.1}$$

We have assumed that, current work load as dedicated tasks for each node, so that the calculation of *makespan* is carried out from the time point when sliding window is selected. Table 4.2 shows that the *makespan* is 73 for the individual in Figure 4.2 with corresponding average utilisation of the HDCS with five computing nodes. Table 4.3 shows that the *makespan* is found to be 78 for the individual in Figure 4.2 with corresponding average utilization (AU) for five computing nodes considering the *current system load* as the initial load.

Every iteration generates an allocation list for the batch of tasks selected using the window. The overloading nodes are prevented by the threshold. Threshold is a value that is used to indicate whether a processor is heavily or lightly loaded and a threshold policy in the load-balancing algorithm reduces the traffic overhead significantly [6]. The central scheduler is updates load information of each computing node. We have used a fixed threshold policy for task allocation to a node. The tasks are assigned to the node only if the threshold has not reached. The threshold for each node is calculated as follows:

$$Threshold = \frac{Number\ of\ acceptable\ nodes}{Total\ number\ of\ nodes\ in\ the\ system} \tag{4.2}$$

4.5 Load balancing using simulated annealing

Simulated annealing is a heuristic method that has been implemented to obtain good solutions for a number of discrete optimization problem [92, 149]. The simulated annealing method mimics the physical process of heating a material and then slowly lowering the

temperature (cooling) to decrease defects so as to minimize the system energy [45, 95]. SA-based scheduling is implemented using an iterative algorithm that only considers one possible solution for each task window at a time. The solution uses similar representation as the fixed window size denoted as $WinSize$. Each iteration selects a batch of tasks from the set of n tasks. Hence the dynamic simulated annealing scheduler operates in batch mode for $\frac{n}{WinSize}$ number of times to allocate n tasks. In the first iteration the scheduler operates on a randomly generated initial solution representing an allocation of a batch of tasks. A new solution is generated based upon the neighbourhood structure [93]. Temperature is used as a control parameter in SA and from a high value decreases gradually with each iteration. This decides the probability of accepting a worse solution at any step and is commonly used as a *stopping criterion*. The initial temperature used is an integer value and decreased by a rate called the annealing schedule [104, 107].

Simulated annealing requires an appropriate representation to find the optimal solution. We have used the fixed length window structure as shown in Figure 4.2. The size of the batch is the maximum number of tasks in the window, also termed as *WinSize* [6, 150]. The use of a linear array helps the index to be used as the task number in the window so that an one dimensional list representation is possible for the solution. The individual element indicates the *node number* on which the corresponding task is to be executed. Each window shows a possible allocation of computing nodes for which the *makespan* can be calculated from the ETC matrix and current load of the nodes in the HDCS. The simulated annealing framework in Algorithm 4.1 uses a fixed number of iterations ξ. Let $F(S)$ be the objective function that computes a fitness value for the initial solution S. Starting from the initial solution, the algorithm computes a new solution S' on each iteration from the neighborhood of the current solution S. The value of the objective function for the two solutions are compared to select the solution to be used for the next iteration. If the current computed solution is worse than the previous solution, it can be accepted with a certain probability with reference to the current temperature.

At each of the iterations, the central scheduler selects a batch of tasks from a task set and allocates it to different computing node, such that the loads of the assigned computing nodes is balanced. This is a well known instance of combinatorial optimization, which is tackled using Algorithm 4.1. With n task to be scheduled on m computing nodes, dynamic SA scheduling routine operates in batch mode with a sliding window of size $WinSize$ to select a batch of tasks from the task queue with the central scheduler. The cooling schedule starts with an initial temperature, T_0, and decreases by a factor $\delta \in (0, 1)$ and takes a constant value for a fixed number of iterations. At the k^{th} iteration, the temperature is set to $T_k \longleftarrow T_0\delta^k$. The cooling process continues for $k = 1, 2, 3, \cdots \xi$ to

Algorithm 4.1 A template for classic simulated annealing algorithm

Require: *initial temperature*, *temperature cooling schedule*, *repetition schedul*
Ensure: *the best computed solution*

```
generate initial solution S
initialize T_0
k ⟵ 0.
repeat
   for all i = 1, 2, ..., ξ do
      generate a solution S' with equal distribution over all possible neighbours
      if (F(S') − F(S)) ≤ 0 then
         S ⟵ S'
      else
         u ⟵ (random number from[0, 1])
         T_k ⟵ T_0 δ^k
         if u < exp (F(S') − F(S)) / T_k then
            S ⟵ S'
         end if
      end if
   end for
   k ⟵ k + 1
until some stopping criterion is met.
```

meet the termination condition. The task schedule TS is generated randomly to allocate the batch of tasks to m machines. In the next iteration a new task schedule $TS^{'}$ can be generated using the move set representation.

4.5.1 Move set generation algorithms

We are presenting algorithms to generate three move set representations namely:*(i) inversion*, *(ii) translation* and *(iii) switching*, for SA. The details of these algorithms are presented as Algorithms 4.2, 4.3 and 4.4 respectively. These are used to produce a new solution S' on each iteration from the neighbourhood of a current solution S.

- **Inversion**
  The inversion process is applied to a task schedule to creates a new task schedule by swapping few positions. Figure 4.3 illustrates the process of inversion of allocation list of 10 tasks on 5 nodes with a *makespan* equal to 109. In this process, we have selected four randomly chosen consecutive positions and replaced them by the

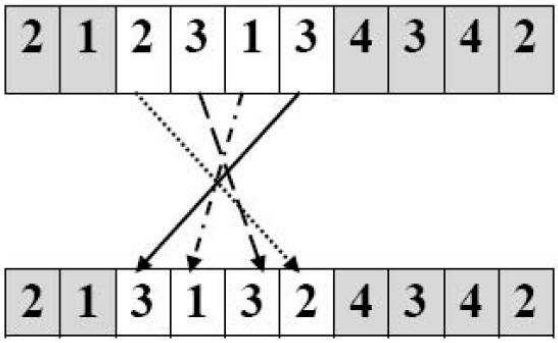

Figure 4.3: The inversion operation

Algorithm 4.2 `INVERSION (TS, WinSize)`

Require: TS $= (ts_1, ts_2, ts_3, \ldots, ts_{10})$: task schedule
$WinSize =$ Size of the TS
Ensure: $TS* = (ts_1, ts_2, ts3, \ldots, ts_{10})$ new task schedule
1: generate a random number $S1$ to represent the starting point and another random number $L1$ for the length of the substring.
2: let $SS = StringReverse(SubString(TS, S1, L1))$;
3: **for** $i = 1$ to $WinSize$ **do**
4: **if** $i < S1$ or ($i > S1$ and $i \geq S1 + L1$) **then**
5: $S = concat(S,\ TS(i))$;
6: **end if**
7: **if** $i == S1$ **then**
8: S = concat (S, SS);
9: **end if**
10: **end for**
11: return (TS);

reverse order of the patterns. This results in a schedule with a *makespan* equal to 82.

- **Translation**
  Translation is a transformation functions that removes two or more consecutive nodes from the schedule and places it in between any two randomly selected consecutive nodes. The translation action performed by Algorithm 4.3 is shown in Figure 4.4. The new schedule also a *makespan* equals to 109.

- **Switching**
  Move set can be constructed for the schedules using a switching function as discussed

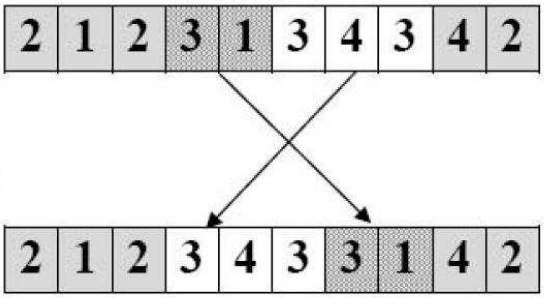

Figure 4.4: The translation operation

Algorithm 4.3 `TRANSLATION (TS, WinSize)`

Require: TS $= (ts_1, ts_2, ts_3, \ldots, ts_{10})$: task schedule
$WinSize =$ size of the TS
Ensure: $TS* = (ts_1, ts_2, ts_3, \ldots, ts_{10})$ new task schedule

```
1: generate a random number S1 to represent the starting point and another random
   number L1 for the length of the substring.
2: generate a random number I1 for the insertion point;
3: let SS = SubString(TS, S1, L1);
4: for i = 1 to WinSize do
5:    if (i < I1) or (i > S1 or ( i > S1 and i ≥ S1 + L1 )) then
6:       S = concat(S, TS(i));
7:    end if
8:    if i == S1 then
9:       S = concat (TS, SS);
10:   end if
11:   if (i > I1) and ((i < S1 or ( i > S1 and i >= S1 + L1 )) ) then
12:      S = concat(S, TS(i));
13:   end if
14: end for
15: return (TS);
```

in Algorithm 4.4, which randomly selects two nodes and switches them in a schedule. Generally speaking, the switching move set tends to rupture the original schedule and results in an allocation that has a *makespan* significantly different from that of the original allocation. Application of Algorithm 4.4 is shown in Figure 4.5. Starting with an initial schedule with a *makespan* of 109, the new schedule generated after switching is 95.

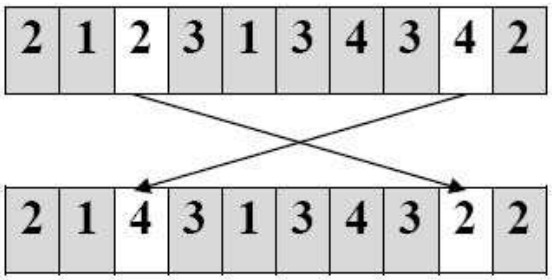

Figure 4.5: The switching operation

Algorithm 4.4 `SWITCHING (TS, WinSize)`

Require: TS $= (ts_1, ts_2, ts_3, \ldots, ts_{10})$: task schedule
$WinSize$ = Size of the TS
Ensure: $TS* = (ts_1, ts_2, ts_3, \ldots, ts_{10})$ new task schedule
1: generate a random number i to represent the task 1 and another random number j to represent task 2.
2: swap(TS(i), TS(j));
3: return (TS);

4.5.2 Simulated annealing framework

The dynamic SA-based load balancer operates in a batch mode to assign n tasks. The i^{th}batch of tasks is denoted as $B[i]$ and the number of tasks in a batch is $|B[i]| = WinSize$. It is assumed that the number of tasks n is integer multiple of $WinSize$. The simulated annealing framework to find the optimal schedule TS for a batch of tasks $B[i]$ is illustrated in Algorithm 4.6. Simulated annealing based dynamic scheduler in Algorithm 4.5 operates in batch mode and selects a batch of tasks $B[i]$ in every iteration. Algorithm 4.6 is executed to produce an optimal schedule for the selected batch of tasks $B[i]$ in a fixed number of iterations. From an initial schedule TS, the simulated annealing approach produces a new schedule using the move set. The move set can be created for an initial schedule, by any one of the three different methods: *Inversion*, *Translation*, *Switching*, through random selection. Four common approaches used as the stopping criteria in simulated annealing algorithm are to use, *(i) a given number of iterations*, *(ii) a time limit*, *(iii) a given number of iteration withouts any improvement in the objective function value*, *(iv) limit on the value of the objective function set by the user* [128, 154]. Our study uses the first approach of fixing the maximum number of iteration as *stopping criteria*.

The implementation of Algorithm 4.6 is based on deciding the values of various param-

Algorithm 4.5 Simulated Annealing Scheduler

Require: $n:$ *number of task*, $m:$ *number of computing node*, *batch size* : $WinSize$
Ensure: L : *makespan*
1: $L_j \longleftarrow 0$ for all nodes
2: **for** $i = 1$ to $n/WinSize$ **do**
3: select a batch of task $B[i]$
4: call Algorithm 4.6: $SADLB(B[i], WinSize)$
5: assign tasks in $B[i]$ to computing nodes as per TS
6: update load of the assigned computing node
7: update expected completion time of unallocated tasks
8: **end for**
9: $L \longleftarrow max_j L_j$
10: return *makespan*: L

eters such as *initial temperature, cooling factor, cooling schedule*, and *stopping criterion*. The simulated annealing scheduler iterates $n/WinSize$ times to allocate n tasks dynamically to m computing nodes in the batch mode. Each iteration invokes Algorithm 4.6 to allocate a batch of tasks to the computing nodes of the system. The major annealing parameter, on which the quality of the final solution depends are the choice of an initial temperature and the choice of a cooling factor. These factors along with the stopping criterion contribute to the success of the SA algorithm [45]. These parameters depends on the application or the problem domain to which it applied. The initial temperature T_0 is set to five to start Algorithm 4.6. The cooling factor δ is a uniform random number selected from the interval $(0, 1)$. Fixed number of iteration is used as the stop criterion for the Algorithm 4.6. We have used a maximum of 40 iteration as *stopping criterion*. The resulting schedules are used to allocate a batch of tasks to computing nodes. On every invocation, Algorithm 4.6 computes a new schedule for the batch of tasks selected and also computes its corresponding *makespan*. The complete execution of Algorithm 4.5 assigns all of the n tasks to the computing nodes and computes the *makespan*. The *makespan* is presented as the *completion time* and the corresponding average processor utilization is computed using Equation 4.1.

4.5.3 Simulation environment and results

The proposed algorithms are coded in Matlab (R2008a) and tested by varying the task pool size from 10 to 1000 on 60 computing nodes. We have compared the SA-based dynamic load balancer (Algorithm 4.5) with immediate mode heuristic load balancing. A randomized resource allocation algorithm has been selected because, *randomized algorithms* are known to give efficient approximate solutions to intractable problems with

Algorithm 4.6 `SADLB (B[i], WinSize)`

Require: *initial temperature*, *temperature cooling schedule*, *repetition schedul*
Ensure: TS with minimum makespan, makespan

randomly generate initial solution TS for a batch of task $B[i]$
initialize T_0
$k \longleftarrow 0$.
repeat
  for all $i = 1, 2, \ldots, \xi$ **do**
    generate a random integer m from the set $1, 2, 3$
    if m = 1 **then**
      $TS' \longleftarrow INVERSION(TS, WinSize)$
    end if
    if m = 2 **then**
      $TS' \longleftarrow TRANSLATION(TS, WinSize)$
    end if
    if m = 3 **then**
      $TS' \longleftarrow SWITCHING(TS, WinSize)$
    end if
    if $\Big(makespan(TS') - makespan(TS)\Big) \leq 0$ **then**
      $TS \longleftarrow TS'$
    else
      $u \longleftarrow (random\ number\ from[0, 1])$
      $T_k \longleftarrow T_0 \delta^k$
      if $u < \exp \dfrac{\Big(makespan(TS') - makespan(TS)\Big)}{T_k}$ **then**
        $TS \longleftarrow TS'$
      end if
    end if
  end for
  $k \longleftarrow k + 1$
until some termination criterion is met.
return TS and $makespan(TS)$

better complexity bounds. Moreover, *randomized algorithms* are selected for performance comparison as these are simple to describe and implement than the deterministic algorithms. We have used immediate mode scheduling algorithms in Section 3.6.1 and 3.6.2, with the task queue as a linear list with the central scheduler. To differentiate with the immediate mode schedulers with priority queue in Algorithm 3.4 and 3.5, we rename the immediate mode scheduler in Algorithm 3.4 and 3.5 with linear queue as FF and RAND respectively.

The simulation results are presented in Figures 4.6 and 4.7 with completion time and

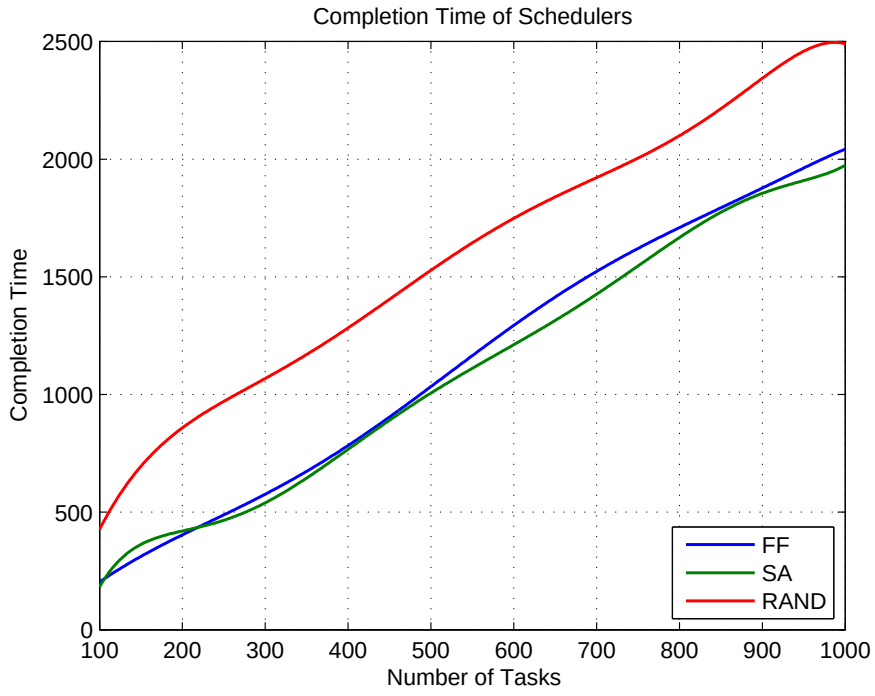

Figure 4.6: Completion Time varying number of task on 60 node

processor utilization respectively. The FF and RAND algorithms for resource allocation can make instantaneous decisions in allocation of a task to the computing nodes, which results in shorter *makespan*. The SA-based load balancing algorithm shows very much similar performance to that of FF in both average *processor utilization* and completion time or *makespan*.

4.6 Load balancing using genetic algorithm

The genetic algorithm is an intelligent optimization and search technique based on the principles of genetics and natural selection [155]. It consists of four main steps, namely initialization, evaluation, exploitation, and exploration [103, 128, 156]. In GA a population is composed of many individuals to evolve under a specified selection rule. Every individual of the population is a solution with its fitness corresponding to the objective function. At each step (iteration) the GA selects individuals at random from the current population to become parents, and uses them to produce their children with the help of genetic operators. Over successive generations, this process evolves toward an optimal solution [128, 154]. The next generation can be created from the current population using

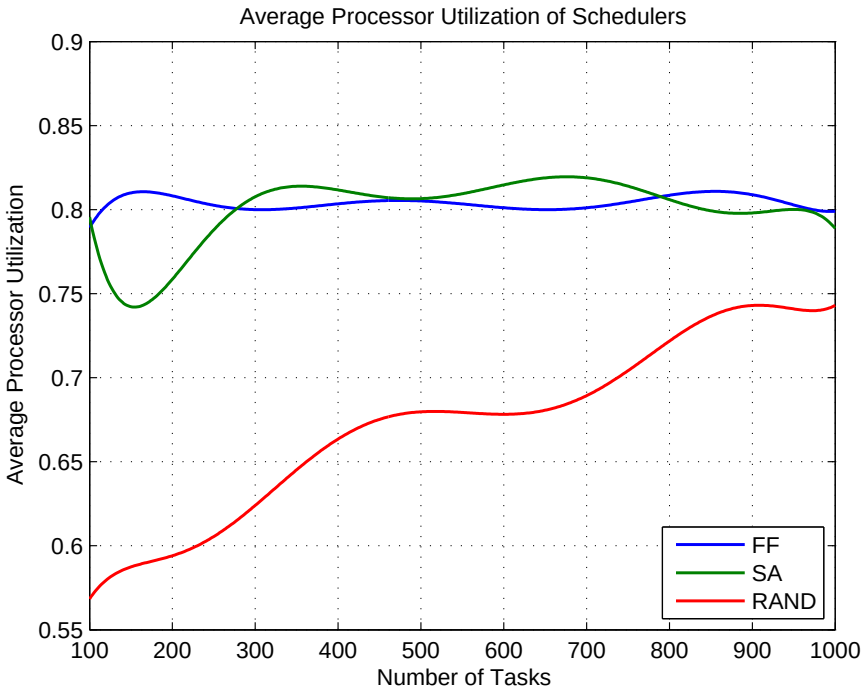

Figure 4.7: Average Processor Utilization varying number of task on 60 node

three main types of rules. These are:

- *The selection rules* selects the individuals, called parents from the mating pool or current population.
- *The crossover rules* combines two individuals otherwise called parents to form the children for the next generation.
- *The mutation rules* makes random changes to the genes of individual parent to form children.

The general template of a simple genetic algorithm consists of abstract steps and is shown as Algorithm 4.7 [103].

The number of genes and their values in each individual are specific to the problem. We have used chromosomes of length equal to the window size, which represents the maximum number of task that fits to the window . The genes in the individual are the node numbers on which the respective tasks to be executed. The initial population is generated randomly corresponding to the batch of tasks selected. The GA operates on a fixed number of tasks and each task is characterized by the task model discussed in

Algorithm 4.7 Simple Genetic Algorithm

Require: *population size*, *cross over probability*, *mutation probability*
Ensure: *the fittest individual*
1: generate initial population
2: calculate the fitness of each individual
3: **while** fitness value ≠ Optimal value **do**
4: Selection; {natural selection, survival of fittest}
5: Crossover; {reproduction, propagate favourable characteristics}
6: Mutation; {apply mutation}
7: **end while**
8: calculate fitness of individuals
9: return fittest individual

Section 2.5 with an integer value to representing the expected computation time. The construction of genetic algorithm for load balancing problem can be divided into four parts: *the representation of individuals* in the population (also termed as the chromosome structure), *the determination of fitness function*, the design of *genetic operators* and the fixing of *probabilities to control genetic operators*. The genetic scheduler operates in an environment where the load status of the computing nodes changes dynamically. It operates in a batch fashion and utilises a GA to minimise the total execution time. Our GA based dynamic load balancing algorithm has been realized using the batch mode heuristic. We have proposed a new codification scheme to represent a task allocation list for a batch of tasks as an individual. We have also introduced different genetic operators that are suitable to this coding scheme. Our proposed GA load balancer operates for a fixed number of iterations to allocate n tasks to m computing nodes.

4.6.1 Chromosome structure

Genetic algorithms require a suitable representation and evaluation mechanism. The proposed GA load balancer are based on fixed length chromosome structure, with integer value assigned to individual genes as the node number. We have used the chromosome structure as shown in Figure 4.2, the length of a chromosome is the maximum number of task in the window and represented as *WinSize* in this thesis [6, 57]. Hence the number of elements in the window is fixed and the length of chromosome is equal to the $WinSize$. The linear array helps to use the index as task number in the window so that an one dimensional chromosome representation is resulted. The individual gene on chromosome indicates the machine on which the corresponding task to be executed. Each chromosome shows a possible allocation of computing nodes for which the *makespan*

can be calculated from the ETC matrix and current load on the computing node. The *makespan* for individual in Figure 4.2 found to be 73 as shown in Table 4.2 along with corresponding *average utilization*. We have simulated the proposed GA load balancer with individual of size 10, however an analysis is also presented to study the performance of proposed GA load balancing scheme by varying length of individual that corresponds to window size.

4.6.2 Fitness function

In genetic algorithm literature, the term *evaluation* and *fitness* are sometimes used interchangeably. In this chapter, *the evaluation function*, or *objective function* provides the measure of *makespan* with respect to dynamic load balancing problem as define in Equation 2.3. The *fitness function* transforms that measure of performance into an allocation of reproductive opportunities [130]. The evaluation of an individual representing a set of parameters is independent of the evaluation of any other individual. The fitness of that individual, however, is always defined with respect to other members of the current population. The fitness function used is based on three performance metric *i) makespan*, *(ii) average utilization*, and *(iii) acceptable queue size*. The GA scheduler proposed in this chapter uses *fitness function* to evaluate the quality of the task assignment for the individual has been adapted from [6],and defined by Equation 4.3.

$$Fitness = \frac{1}{makespan} \times AU \times \frac{\#\,acceptable\ queues}{\#\,computing\ nodes}, \tag{4.3}$$

where AU is average utilisation. This fitness function is used by Algorithm 4.9 to measure the quality of the task allocation for a selected batch of task on each iteration.

4.6.3 Genetic operators

The basic implementation of GA load balancer follows the Simple Genetic Algorithm(SGA) framework suggested by Goldberg [103, 128]. The execution of GA load balancer is a two stage process. The process begins with the randomly generated *initial population* or *current population*. The selection process is applied to the *initial population* to create a *mating pool* or *intermediate population*. Then the members of *intermediate population* are subjected to *recombination* and *mutation* to create the next population. This process of transforming *current population* to *next population* constitutes one generation. GAs are blind search techniques and hence require problem-specific genetic operators to get the good solutions. The genetic operator used by us to design genetic algorithm scheduler are explained details in subsequent subsections.

4.6.3.1 Selection for reproduction

The reproduction process is used to create a new population of individuals from old population by selecting individuals from old population based on their fitness values. The most common selection schemes used in GAs are (i) *rank selection*, and (ii) *roulette wheel selection* [103, 128, 157]. The reproduction process forms new population, by selecting individuals from the old population based upon their fitness value that optimizes the objective function. Starting with the initial population, parent chromosomes are selected to form a mating pool via proportional selection process, also termed as *"roulette wheel selection"* [104, 128]. This process can view the population as mapping onto a roulette wheel, where each individual is represented by a space that proportionally corresponds to its fitness. By repeatedly spinning the roulette wheel, individuals are chosen using *stochastic sampling with replacement to fill the intermediate population* [130]. The proposed GA load balancer uses *roulette wheel selection* to design new population from a current population.

4.6.3.2 Crossover

Creation of new individuals are performed through *crossover* and *mutation*. The crossover operator is mainly responsible for search aspect of genetic algorithms [135]. On completion of the construction of *mating pool* are subjected to recombination that creates the *next population*. The recombination can occur with the application of crossover to randomly paired individuals with a probability namely *cross over probability* denoted as p_c. Crossover operation selects a pair of individual from the mating pool, then randomly selects two points to apply standard two point crossover, and produces two offspring. GA load balancer in this chapter operates with *crossover probability*; $p_c = 0.7$. Example in Figure 4.8 depicts two point crossover processes with *P1* and *P2* as parent.

In this example two parents *P1* and *P2* are selected randomly with *makespan* value as *109* and *80* respectively based on ETC matrix given in Table 4.1. Two crossover points are randomly selected as 2 and 6 to produce upspring *C1* and *C2* with *makespan* equals to *60* and *103* respectively. The load balancing problem being the minimization problem, two individuals with higher *makespan* are discarded to maintain a constant population size throughout the solution finding process using GA.

4.6.3.3 Mutation

In the process of mutation, the individual is changed by swapping two genes position randomly with a small probability. After crossover, we can apply a mutation operator.

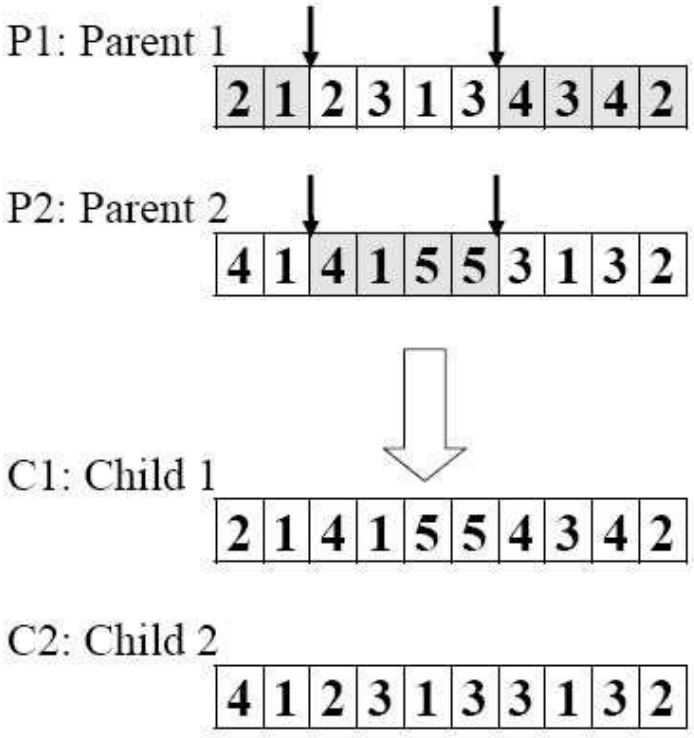

Figure 4.8: Results of two point crossover

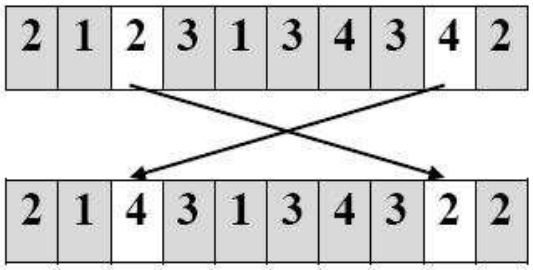

Figure 4.9: Results of mutation on chromosome

For each bit in the individual mutate with some probability known as *mutation probability* and denoted as p_m. Typically the mutation rate is applied with less than 0.15 probability [130]. The mutation probability is used to select the individual that is subjected to

mutation process. The Figure 4.9 depicts the mutation process. Two random positions are selected to exchange their values that designate the node or processor number. This mutation process produces a chromosome with *makespan* as *95* from the chromosome with *makespan* equal to *109* as depicted in Figure 4.9. In a single generation the process of *selection*, *crossover* and *mutation* are applied to the initial population to create the next population.

4.6.4 Generational changes

The process of evaluation, selection, recombination and mutation forms one *generation* in the execution of a genetic algorithm. As we are working on a minimization problem the value of *makespan* in current generation must be less than the *makespan* obtained in previous generation. A generational change must be as per the objective function. An *average makespan* can also be used to justify the progress of iteration to optimize the objective function.

4.6.5 Stopping conditions

Stopping conditions are used to halt the evolution of population. Task is to be assigned on the fly, and the search on the solution space is carried out in random, hence we have to accept the suboptimal solution which can be found at the earliest. The GA evolves the population until it meets one or more stopping conditions. For dynamic load balancing problem we can use two stopping criteria, first the individual with the lowest *makespan* is selected after each generation and if it is greater than a specified minimum or *makespan* computed previous generation. Second for a maximum number of generation fixed as per the number of task to be allocated to HDCS. In our approach, the individual with the smallest *makespan* is selected after each generation. If the *makespan* value of current generation is less than the previous generation, the iteration continues till the maximum generation [141]. If the makespan value found to be higher then the GA stops evolving.

4.6.6 Genetic algorithm framework

The dynamic load balancing algorithm for HDCS uses the *state-of-art* homogeneous GA scheduler by Zomaya *et al.* [6]. The queue with central scheduler contains n number of unscheduled task. A batch of task from the waiting queue with central scheduler is selected in each iteration. The best possible allocation of the batch of task can be found using the Algorithm 4.8. The resulted task schedule by Algorithm 4.8 is used to allocate this batch of task to different computing nodes in the system.

Algorithm 4.8 `GADLB (TS, WinSize)`

Require: population size, crossover probability, mutation probability, batch size: *WinSize*, ETC: expected time to compute

Ensure: individual with minimum *makespan*: TS

1: random generation of initial population for batch of task $B[i]$
2: **for** $i = 1$ to maximum generation **do**
3: evaluation of fitness of individuals (i.e. makespan of each chromosome) of the current population
4: select the new population using roulette wheel method
5: select individuals with crossover probability to apply two point crossover
6: select individuals with mutation probability to apply mutation
7: **end for**
8: return individual with minimum *makespan* as TS

Algorithm 4.9 Genetic Algorithm Scheduler

Require: $n : numberoftask, m : numberofcomputingnode, L : makespan, WinSize : batchsize$

Ensure: $L : makespan$

1: $L_j \longleftarrow 0$ for all nodes
2: **for** $i = 1$ to $n/WinSize$ **do**
3: select a batch of task $B[i]$
4: call Algorithm 4.8: $GADLB(B[i], WinSize)$
5: assign tasks in $B[i]$ to computing nodes as per TS
6: update load of the assigned computing node
7: update expected completion time of unallocated tasks
8: **end for**
9: $L \longleftarrow max_j L_j$
10: return *makespan*: L

The heterogeneous GA scheduler in Algorithm 4.9 operates for fixed number of iteration to allocate n task in batch mode using Algorithm 4.8. The number of task to be allocated are assumed to be integer multiple of batch size ($WinSize$). The Algorithm 4.9 stops after $\frac{n}{WinSize}$ and computes *makespan* for n tasks. The GA load balancer operates on the finite population of chromosomes. The initial population in this problem is based upon the chromosome structure depicted in Figure 4.2. A population size of 20 is used for the fixed window size 10 for maximum number of 40 generation.

Table 4.4: The Process of designing mating pool from initial population

Sl.No	Initial Population	Load on machine	Max load: makespan	$L_i/\sum L$	$\frac{L_i}{AverageLoad}$	Mating pool	Individual
1	2123134342	(18,109,95,21,0)	109	0.114	0.919	2123134342	I1
2	4141553132	(80,5,31,39,31)	80	0.084	0.674	4141553132	I2
3	3322122524	(11,209,11,7,3)	209	0.220	1.762	4141553132	I3
4	3435423531	(2,31,82,35,56)	82	0.086	0.691	3435423531	I4
5	5522423311	(15,170,40,7,60)	170	0.179	1.433	3435423531	I5
6	2234421355	(24,98,68,49,40)	98	0.103	0.826	2234421355	I6
7	4323142254	(11,111,31,48,34)	111	0.116	0.935	4323142254	I7
8	4345331554	(24,0,65,46,90)	90	0.094	0.758	4345331554	I8
		Sum	949	0.996	**7.998**		
		Minimum	80	0.084	0.691		
		Average	118.6≈119	0.124	**0.999≈1**		

Table 4.5: Parameters used in GA for load balancing

Parameters of GA	value
Number of task	1000
Number of node	60
Max. generation	40
Population size	20
Multi point Crossover Rate	0.7
Mutation Rate	0.05
Window size	10

Let us assume that expected completion time of task uniformly distributed in the interval of $[mintime, maxtime]$. $WinSize$ represents maximum number of task or genes in the chromosome. The *inipopsize* is the size of initial population representing chromosomes of equal length. We have used fixed population size for all the iteration with maximum number of generation as stopping criteria. Initial Population (IP) can be generated as an array of size = *inipopsize* using the following formula:

$$IP = mintime + (maxtime - mintime) * rand(inipopsize, winSize = maxtask) \tag{4.4}$$

The procedure to create mating pool from a *initial population* is presented as an example in Table 4.4, where an initial population of size 8 is created using Equation 4.4. The makespan for each member of initial population are computed using ETC matrix shown as Table 4.1. Roulette wheel selection procedure is used to weight the individuals. As load balancing problem is a minimization problem, *third* and *fifth* individual with higher *makespan* are replaced with *second* and *fourth* individual to create the mating pool. The mating pool is also knows as *intermediate population*. Through the above process initial population are subjected for evaluation and selects chromosomes to design intermediate population. The proposed GA load balancer uses fitness function defined in Equation 4.3. Dynamic GA load balancer operates in batch mode to assign n tasks. The i^{th}batch of task is denoted as $B[i]$ and the number of task in a batch is $|B[i]| = WinSize$. GA based dynamic resource allocation, Algorithm 4.9 starts with generating initial schedule TS randomly for a batch of task $B[i]$. Algorithm 4.8 is executed to produce an optimal schedule for the selected $B[i]$ in fixed number of iteration using simple GA. A final allocation list for the batch of task $B[i]$ is obtained after the 40 generation. Our implementation uses a batch of size 10. A batch of task allocation to computing nodes are followed by the load update for every computing node. Hence *expected completion time* for all unallocated tasks are computed, and the GA load balancer uses these updated *expected completion time* values for unallocated task. We simulated the performance of

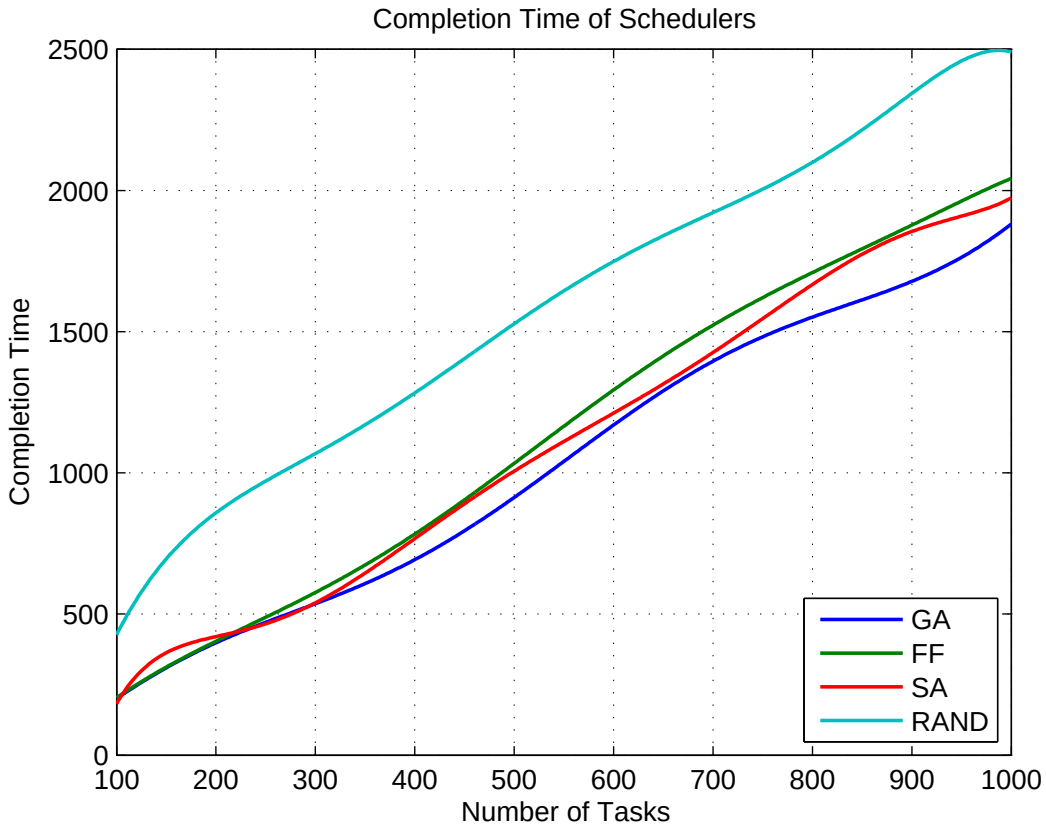

Figure 4.10: Completion time varying task size with central scheduler

our GA and SA scheduler against two immediate mode scheduler.

4.6.7 Experiments and results

The proposed algorithms are coded in Matlab(R2008a) and tested using different set of task pool of size from 100 to 1000. In particular the genetic algorithm for load balancing uses parameters as listed in Table 4.5. We have simulated twenty times for an instance of task size to compute the completion time or *makespan* for a fixed size of task pool. Tasks are submitted to central scheduler using Poisson distribution with arrival rate λ equals to 10. The parameters in Table 4.5 are used to realized simulation using genetic algorithm. For performance analysis four algorithms are considered: proposed algorithm using GA and SA, First Fit(FF) and Radomized(RAND) algorithm. Each iteration selects a set of 10 tasks using a sliding window technique [6]. Iteration updates the window by selecting new tasks from the task queue of central scheduler. These new set of tasks are to be

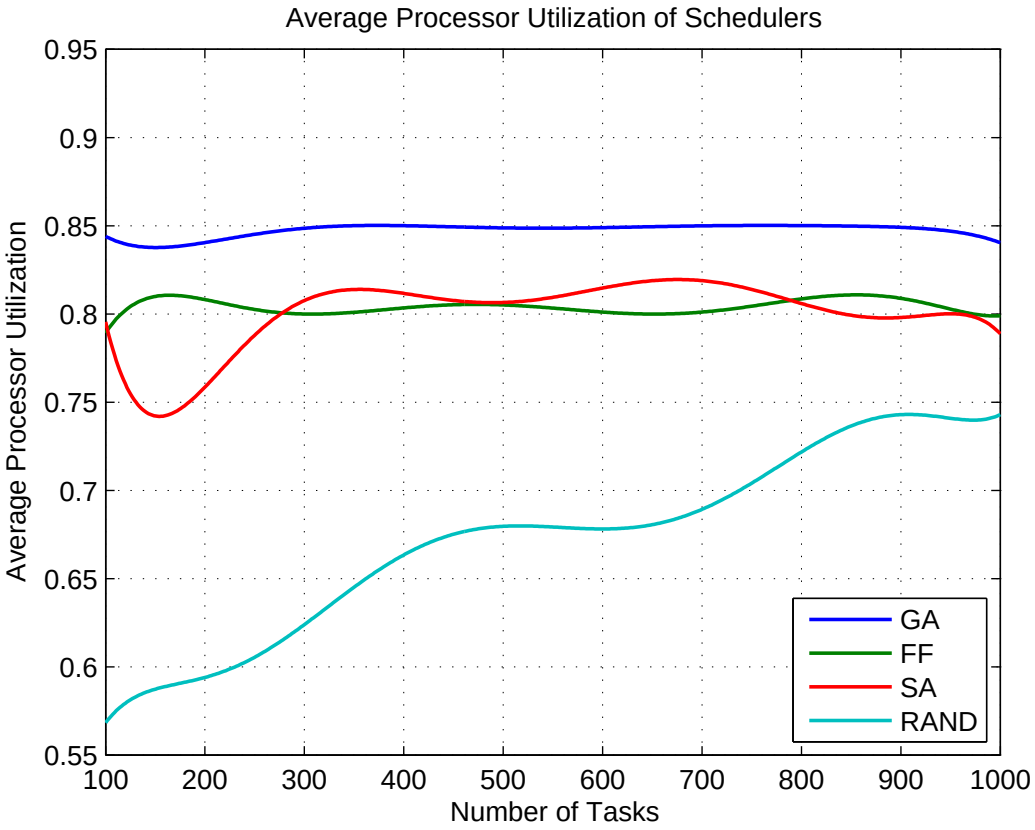

Figure 4.11: Average Processor Utilization varying task size with central scheduler

assigned to the computing nodes in the iteration. These processes are repeated for a fixed number of iteration for GA and SA. Finally an individual with minimum *makespan* is selected as solution and tasks in the window allocated to the computing nodes for execution. We have compared batch mode resource allocation Algorithms 4.9 and 4.5 with immediate mode heuristic load balancing Algorithms 3.4 and 3.5 as discussed in Chapter 3. The immediate mode task allocation algorithms considers single task for scheduling. We refer the FCFS heuristic Algorithm 3.4 as **FF** and random task allocation Algorithm 3.5 as **RAND**.

Performance of genetic algorithm based load balancing scheme is studied by varying the window size from 10 to 50 on fixed number computing nodes $m = 60$ for 100 task. The simulation result presented in Figure 4.12 indicates a decrease in completion time for larger window size. These findings can be useful in selecting appropriate window size for the task pool to meet specific performance requirement. Further simulation study for average processor utilization is depicted in Figure 4.13 by varying the window size.

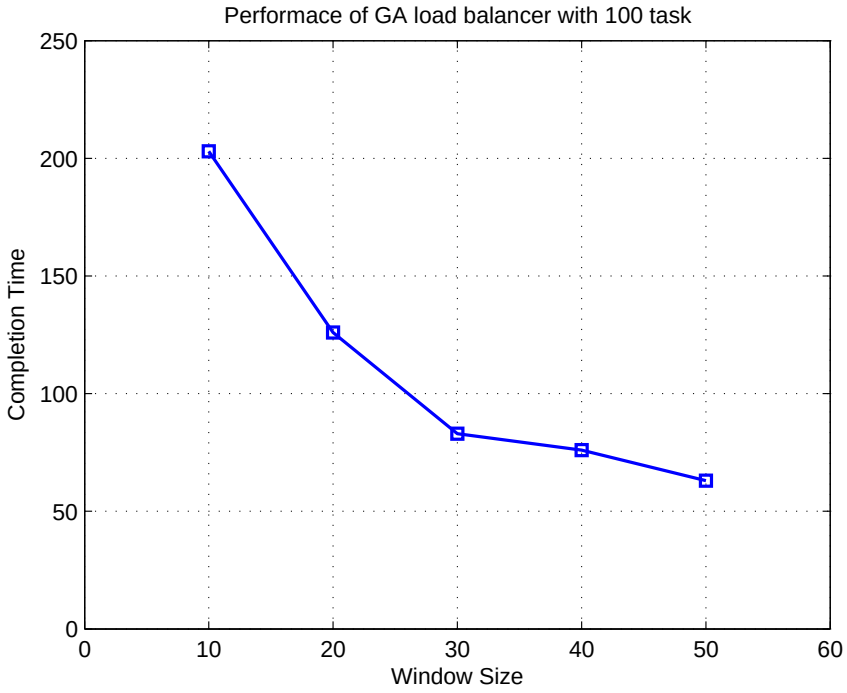

Figure 4.12: Completion time as a function of window size.

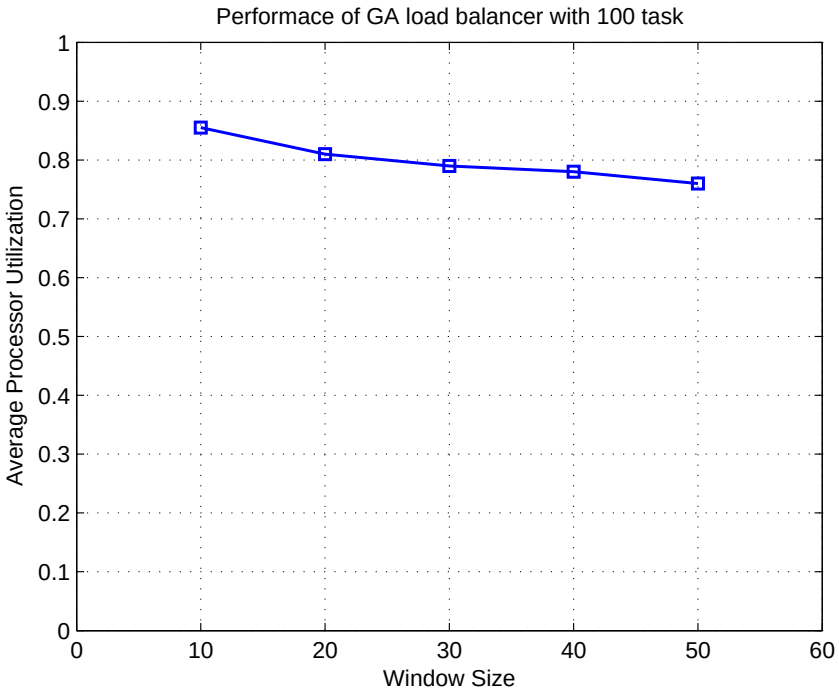

Figure 4.13: Average processor utilization as a function of window size.

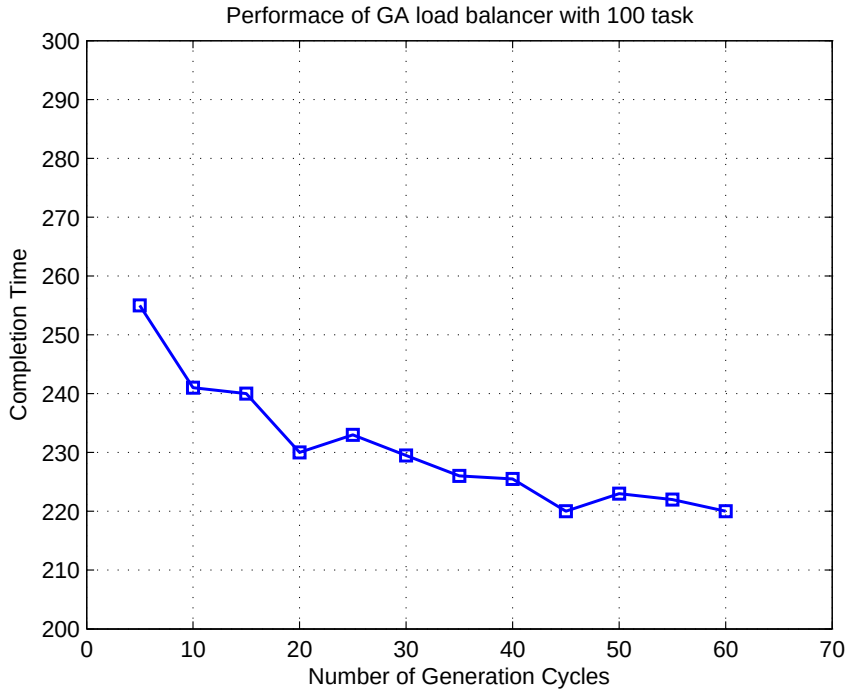

Figure 4.14: Task completion time as function of generation.

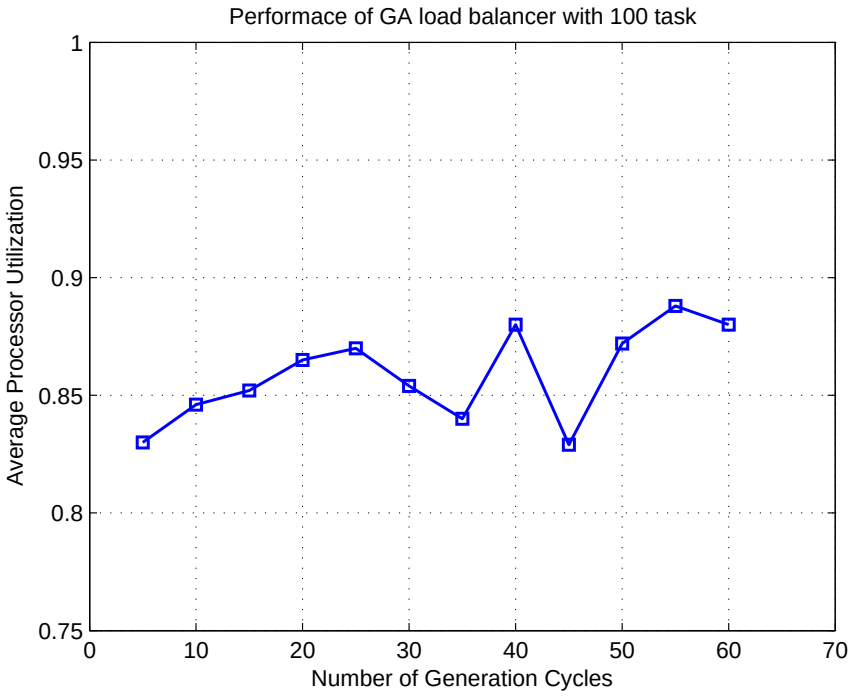

Figure 4.15: Average processor utilization as a function of generation.

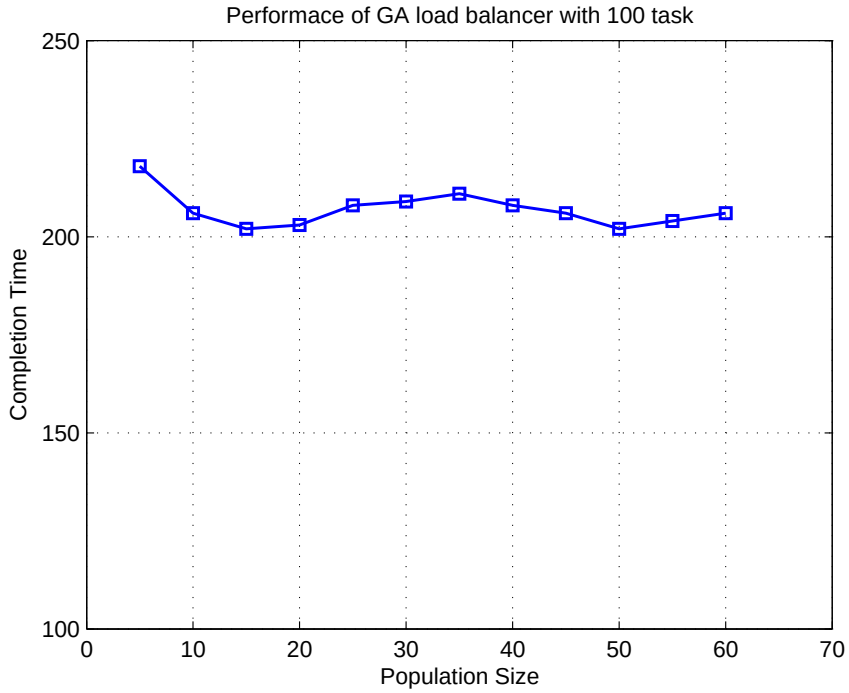

Figure 4.16: Completion time as a function of Population size.

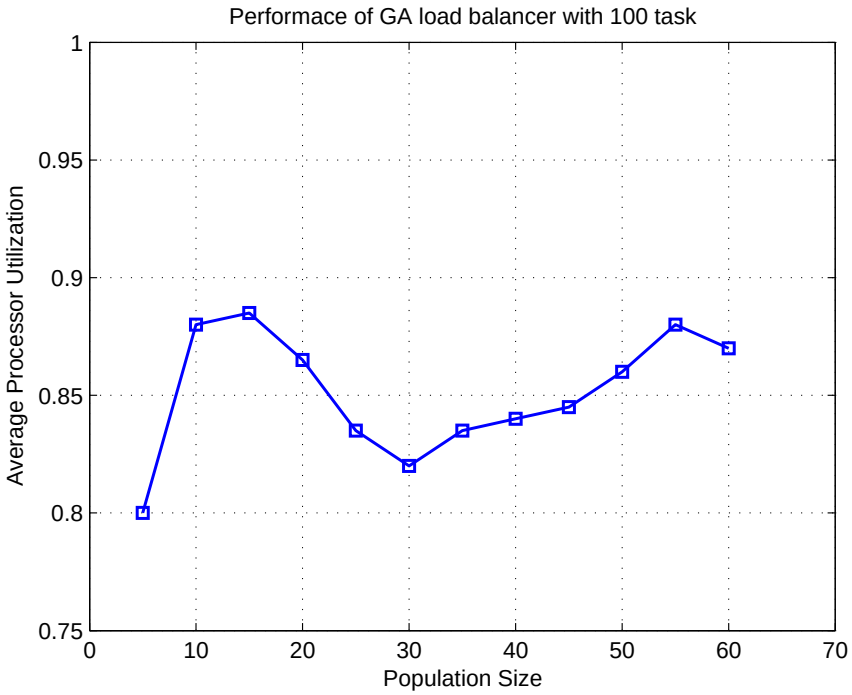

Figure 4.17: Average processor utilization as a function of population size.

The simulation output shows that average processor utilization deteriorates with the increase in window size. Fixed number of generation is being used as one of the stopping criteria for genetic algorithms. The total completion time decreases significantly with the increase in the number of generation from 5 to 60 in Figure 4.14. However after 30 generation no significant performance improvement has been observed. Figure 4.15 shows the average processor utilization with varying number of generation. The result presented also indicates the optimum value of processor utilization for the generation.

Different population size has been selected to study the performance of genetic algorithm for load balancing. The Figure 4.16 indicates a decrease in completion time for the tasks with the increase in population size. This study may be use to fine tune the genetic algorithm for a desired performance.

The average processor utilization with changing population size is shown in Figure 4.17. The result shows the optimum processor utilization, but the performance decreases with increase in population size. The simulation results presented here are for the randomly generated ETC matrix.

4.7 Conclusion

We have designed and tested schedulers based on the GA and SA. We have introduced a suitable codification for GA and SA for dynamic load balancing on the HDCS. The Fast come first served or the first fit (FF) and randomized algorithms for resource allocation can make an instantaneous decision to allocate tasks to computing nodes, which results in a shorter *makespan*. The proposed dynamic task allocation algorithms use the sliding window techniques to select a batch of tasks, and allocates them to the computing nodes in the HDCS. The proposed GA-based dynamic load balancer has been found to be effective, especially in the case of a large number of tasks. This load balancer worked rather well in terms of achieving the goals of minimum total completion time and maximum processor utilization.

Chapter 5

Approximation Algorithms for Load Balancing

Approximation algorithms have been used to design polynomial time algorithms for intractable problems that provide solutions within the bounded proximity of the optimal solution. Load balancing algorithms attempt to compute the task assignment with the smallest possible makespan*. This chapter presents an analysis and design of approximation algorithms based on task and machine heterogeneity through the ETC matrix on an HDCS with* makespan *as the performance metric. The proposed approximation scheme has been compared with an optimal solution computed as a lower bound.*

5.1 Introduction

It is unlikely to have exact algorithms for NP-hard problems. One has to agree for sub-optimal solutions that can be found out in polynomial time [33, 86]. The load balancing problem is a minimization problem, with the objective of minimizing the *makespan* of n tasks on m computing nodes [6, 86, 87]. The problem of finding a solution to the load balancing problem, defined in Equation 2.3, is NP-hard [33]. An optimization problem is NP-hard, if the associated decision problem is NP-complete. The load balancing problem can be proved to be NP-hard by reduction from the *partition problem* [86]. NP-hard problems are intractable, which means that there does not exist an efficient algorithm that is guaranteed to find an optimal solution for such problems. Approximation algorithms find solutions that are guaranteed to be close to optimal or sub-optimal in polynomial time. The solution produced by approximation algorithms are to be compared with the

optimal solution of the problem. The optimal solution to minimization problems can derived as a lower bound and for maximization problems it is an upper bound. Load balancing problems are presented in this thesis as minimization problems; hence defining optimal solutions to the problem can found out theoretically with the help of lower bound theory [86, 158].

Many real world optimization problems are NP-hard, and it is desirable to solve large instances of these problems in a reasonable amount of time [110]. Sometimes, we have a polynomial time algorithm for a problem P; however, the time complexity of this algorithm becomes super polynomial while solving problems with large instances. In practice, it is desirable to have an algorithm with polynomial time complexity to deal with large instances. The most common approach used by the researchers to find solutions to NP-hard problems were treating them with *integer programming tools* or *heuristics* or *approximation algorithms* [86, 87]. Heuristic algorithms can produce the result quickly on a large instance provided the heuristic is able to deal with that instance. In general, heuristics based approaches do not work effectively on all problem instances [110]. Heuristic algorithms may produce good solutions against the quality of the solution. Whereas approximation algorithms have the capability to produce solutions, which are guaranteed to be within some constant bound of the optimal solution.

Approximation algorithms generally have two properties [159]. They provides a feasible solution to a problem instance in polynomial time, and also ensures some quality of the solution. The quality of an approximation algorithm is the maximum *distance* between its solution and the optimal solution, evaluated for all possible instances of the problem. An approximation algorithm is characterised by a factor ρ called *the approximation factor* or *approximation ratio*; for some $\rho < 1$ and it is named as a $\rho-approximation\,algorithm$ [100, 160].

Let A be an polynomial time algorithm that produces a feasible solution to every instance I of a problem P. Let the value of the optimal solution to the problem instance I be denoted as $FO(I)$ and let the $FA(I)$ be the feasible solution produced by an Algorithm A on the same instance I. Then, the following definitions are used to characterize approximation algorithms:

Definition 5.1 (Approximation algorithm)**.** *An approximation algorithm for a problem P is an algorithm that produces approximate solutions for P.*

Definition 5.2 (Absolute approximation algorithm)**.** *An algorithm A is said to be an absolute approximation algorithm for a problem P if and only if, for every instance I of P,*

$$|FO(I) - FA(I)| \leq k, for\ some\ constant\ k.$$

Definition 5.3 ($\rho-$ approximation algorithm)**.** *A is $\rho -$ approximation algorithm if and only if, for every instance I of size n,*

$$|FO(I) - FA(I)\ |\ /FO(I) \leq \rho$$

for some constant ρ. $FO(I)$ is assumed to be greater than zero.

A $\rho -$ *approximation algorithm* is guaranteed to produce a solution with objective function value at most ρ times the optimal solution [161]. The most desirable kind of approximation algorithms are *absolute approximation algorithms* [110]. The approximation algorithms for dynamic load balancing discussed in this chapter are proved to be *absolute approximation algorithms*.

To prove an *algorithm* to be a $\rho -$ *approximation algorithm* for the problem P, it is required to know the optimal solution to the problem P. Since the optimal solution to the *load balancing* problem in an HDCS is not known, lower bound of the problem is to be used to characterize the proposed approximation algorithms. Here, load balancing is a job scheduling policy which takes a job as a whole and assign it to the computing node [5].

5.2 Related work

An algorithmic approach to the load balancing problem is presented in [86]. The algorithmic approaches used for solving load balancing problem are roughly classified as (i) exact algorithms, (ii) heuristic algorithms, and (iii) approximation algorithm [87, 99]. Fundamental load balancing problem is used by Kleinberg and Tardos [86] to illustrate some of the basic issues related to the design of approximation algorithms.

A simple family of approximation algorithms for solving the generalized assignment problem has been presented by Cohen *et al.* [162] using "local-ratio technique". A review is presented in [163] considering the ten most open questions in the area of polynomial time approximation algorithms for *NP-hard* scheduling problems. Approximation algorithm for scheduling of n jobs on m identical machines has been presented by Graham [87]. A polynomial-time 2-approximation algorithm was presented by Shmoys and Tardos [164], that minimized the *makespan of the schedule* and the *mean job completion time* for the generalized assignment problem for n independent tasks on m heterogeneous machines. A polynomial time 2-approximation algorithm for the single criterion problem of minimizing the *makespan* was given by Lenstra *et al.* [161]. Fast approximation

algorithms for resource allocation is suggested in [165] and applied to large linear programming problems with packing and covering constraints. Alon *et al*. [166] presented an $\varepsilon-$approximation scheme for the general *load balancing* problem on m identical machines. An efficient approximation algorithm for solving the generalized assignment problems presented in [162] with a $(1+\alpha)$ approximation ratio, where α is the approximation ratio for the knapsack algorithm. Chen and Choi [167] presented a 2-approximation algorithm for data distribution with load balancing in web servers. An improved *(1/3)-approximation algorithm* for resource allocation is presented in [168] for reusable resources for the set of n tasks. An efficient approximation algorithm for load balancing with resource migration in distributed systems is suggested in [169], by partitioning the system into regions. Chudak and Shmoys [170] presented an $\bigcirc(log\, m)$ approximation algorithm for n jobs on m heterogeneous machines. They have also formulated a linear programming problem using the speed at which a job is to be processed on the computing nodes.

5.3 Proposed approximation schemes

Approximation algorithms are being used to tackle *NP-hard* optimization problems. Commonly four general techniques are used to design approximation algorithms: *(i) Greedy algorithms*, *(ii)Primal-dual technique*, *(iii)Linear programming and rounding*, and *(iv)Dynamic programming* [86, 160]. Algorithms for optimization problems typically go through a sequence of steps, with a fixed set of choices for each step. A greedy algorithm always makes a choice that is locally optimal in the hope that it will lead to a globally optimal solution. The proposed approximation algorithms are based on the HDCS model presented in Chapter 2. We have considered a heterogeneous distributed computing system with a set of $M = \{M_1, M_2, ..., M_m\}$, of m independent heterogeneous computing nodes as shown in Figure 2.1. The dynamic load balancing problem, presented as a linear programming problem (LPP) in Section 2.6, is restated below for ready reference.

$$Minimize\, L = \sum_{j=1}^{m} x_{ij} = t_{ij},\, \forall\, t_i \in T \tag{5.1}$$

subjected to:

$$\sum_{j=1}^{n} x_{ij} \leq L,\, \forall\, M_j \in M$$

$$where\, x_{ij} \in \{0, t_{ij}\}, \forall t_i\ \in T, and\, M_j \in M$$

$$x_{ij} = 0,\, \forall\, t_i \notin A(j)$$

The approximation algorithm design is concerned with allocating a set of n tasks to these computing nodes. Let T be the set of tasks; that is, $T = \{t_1, t_2, ..., t_n\}$. Each task t_i has an expected time to compute denoted as t_{ij} on node M_j. The tasks are arriving from different nodes to the central scheduler or serial scheduler. The tasks have an equal probability of being allocated to any of the m computing nodes.

Let $MaxTask$ be the maximum number of task in the HDCS. Let $MaxNode$ be the number of computing nodes in the HDCS. The task to be executed on the HDCS can be represented as an ETC matrix of dimension $MaxTask \times MaxNode$. This matrix representation is also known as consistent ETC matrix as discussed in Section 2.5. A simple load balancing approximation algorithm for the HDCS based on the greedy paradigm is shown in Algorithm 5.1. The tasks are assigned one by one, to the computing nodes by selecting the node with the minimum load at each step. Selection of the minimum load from the m nodes can be done in $\bigcirc(1)$ time by using a binary min-heap. A min-heap with m nodes can be used to maintain the current load of m computing nodes in the HDCS. The heap can be updated in $\bigcirc(log\, m)$ time for each T_j with machine M_j. The running time of Algorithm 5.1 is $\bigcirc(n\, log\, m)$ for assignment of n number of tasks.

Algorithm 5.1 `Greedy resource allocation`

Require: $ETC(MaxTask, MaxNode)$
Ensure: $T : makespan$
1: $T_j \longleftarrow 0\ forall$ node M_j
2: $A(j) \longleftarrow \phi\ forall$ node M_j
3: **for** $i = 1$ to MaxTask **do**
4: Let M_j be a node with minimum T_j
5: Allocate task t_i to Node M_j
6: $A(j) \longleftarrow A(j) \cup \{t_i\}$
7: $T_j \longleftarrow T_j + t_{ij}$
8: **end for**
9: $T \longleftarrow max_j T_j$

This algorithm assumes the initial load of the node(machines) to be zero. The algorithm stops only when all tasks are assigned and complete their execution on the m computing nodes. The solution to the load balancing problem is reported as the *makespan*, which has been calculated from a valid allocation of tasks to different computing nodes.

A lower bound has been used as an estimate of the minimum amount of work needed to solve a given problem. Deriving good lower bounds is often more difficult than devising efficient algorithms, because it is not possible to analyze and enumerate all the

possible algorithms. The lower bound can be established in four different way: *(i) trivial lower bounds*, *(ii) information-theoretic arguments (decision trees)*, *(iii) adversary arguments*, and *(iv) problem reduction* [158]. The lower bound to dynamic load balancing problem is obtained through *trivial lower bound* . The lower bound of the minimization problem in Equation 5.1 can be calculated with the observation that, if it is possible to allocate the tasks over all the m computing nodes equally, the load on each node will be $(\sum_{1\leq j\leq m} L_j)/m$. Moreover, if a task is assigned to the slowest machine M_m, the completion time of that task can be decisive for the lower bound. The lower bound can be obtained as, the maximum time taken by a task to complete the processing on node M_m and computed as $max_{1\leq i\leq n} t_{im}$. Let L_{max} denotes the optimal solution for the load balancing problem, then the following equation holds for the HDCS with m computing nodes:

$$L_{max} \geq max\,(\,(\sum_{1\leq j\leq m} L_j)/m,\; max_{1\leq i\leq n} t_{im}) \tag{5.2}$$

Hence, the lower bound for the load balancing problem, denoted as L_{min}, is defined as

$$L_{min} \;=\; max\,(\,(\sum_{1\leq j\leq m} L_j)/m,\; max_{1\leq i\leq n} t_{im}) \tag{5.3}$$

This lower bound computed for the task allocation on the HDCS is used to characterize the proposed approximation algorithms in Section 5.3.1 and 5.3.2.

5.3.1 2-approximation algorithm for load balancing

Theorem 5.3.1. *The Greedy resource allocation is a 2-approximation algorithm.*

Proof. Let L_{max} be the optimal solution to the load balancing problem on the HDCS. It is to be proved that *Greedy resource allocation* always completes an assignment of n tasks to the computing nodes such that the makespan T is less than or equal to L_{max}. Let M_k be the machine with load $A(k)$ that determines the *makespan* for the set of task represented as ETC matrix using Algorithm 5.1 on m computing nodes. On successful execution of Algorithm 5.1, we have $T_k = max_{1\leq j\leq m} T_j$. Let t_{i*} be the last task that is assigned to node M_k, where at the time of assignment of t_{i*}, the computing node M_k is with minimum load among all of the m nodes. Let $T_k^{'}$ be the load of machine M_k just before the assignment of task t_{i*}, then $T_k = T_k^{'} + t_{i*k}$ and $T_k^{'} \leq T_j^{'}\; for\; all\; 1 \leq j \leq m$. This leads to the following equation:

$$mT_k^{'} \leq \sum_{1 \leq j \leq m} T_j^{'} = \sum_{1 \leq i < i*} t_{ik} < \sum_{1 \leq i \leq n} t_{ik} \leq mL_{min} \tag{5.4}$$

As $T_k^{'} < L_{min}$, we can have

$$T_k = T_k^{'} + t_{i*k} \leq L_{min} + t_{i*k}$$

As $t_{i*k} \leq max_{1 \leq i \leq n} t_{im}$

$$T_k \leq L_{min} + max_{1 \leq i \leq n} t_{im}$$

Using Equation 5.2

$$T_k \leq 2.L_{max}$$

□

The greedy resource allocation is never more than a factor 2 from optimal solution for load balancing problem as proved in Theorem 5.3.1. This *resource allocation* leads to a larger *makespa*n, when we have a large number of tasks, each with a small expected time to compute, followed by a single very large task. Then Algorithm 5.1 will assign small tasks evenly on the computing nodes followed by the large task to one of the nodes. A better allocation is possible by assigning the large task to the machine with the least ETC value, followed by allocation of small tasks among the rest $m-1$ nodes. Hence, a better greedy algorithm is possible by using the ETC matrix, where the tasks are arranged according to increasing value of the expected time to compute on the m computing nodes.

5.3.2 3/2-approximation algorithm for load balancing

Let the computing node M_1 be the fastest computing node and M_m be the slowest computing node in the HDCS. We assume that the service times follow exponential distribution with node M_j having service rate μ_j. Let the computing nodes be arranged in descending order of their service rates. This results in a consistent ETC matrix for the n number tasks on m nodes, so that $t_{ij} \leq t_{ik}$ for task t_i on machine M_j and M_k, with

Table 5.1: Sorted Expected Time to Compute:SETC

$Task/Node$	M_m	M_{m-1}	$\cdots$	M_1
t_1	t_{1m}	$t_{1(m-1)}$	$\cdots$	t_{11}
t_2	t_{2m}	$t_{2(m-1)}$	$\cdots$	t_{21}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_n	t_{nm}	$t_{n(m-1)}$	$\cdots$	t_{n1}

$\mu_j \geq \mu_k$. The resulting ETC matrix, denoted as $SETC$ is shown in Table 5.1. For an arbitrary task t_i, we have $t_{i1} \leq t_{i2} \leq \ldots \leq t_{im}$.

Lemma 5.3.2. *Let $T = \{t_1, t_2, ..., t_n\}$ be the set of n tasks with each task t_i having an expected time to compute t_{ij} on node M_j. If T can be scheduled on m machines, where $t_{im} \geq t_{i(m-1)} \geq \cdots \geq t_{i1}$, then $L_{max} \geq t_{m1} + t_{(m+1)1}$.*

Proof. Suppose there are $m+1$ tasks to be assigned to m heterogeneous machines, then at least two of the task from $t_1, t_2, ..., t_m, t_{m+1}$ are to be assigned to the same machine. As M_1 is the fastest machine, if those two task are to be assigned to the fastest machine, then load of the machine can be at least $t_{m1} + t_{(m+1)1}$. Hence, the *makespan* of the system be $L_{max} \geq t_{m1} + t_{(m+1)1}$. □

Algorithm 5.2 `Sorted Greedy resource allocation`

Require: $SETC(MaxTask, MaxNode)$
Ensure: $T : makespan$
1: $T_j \longleftarrow 0\ forall$ node M_j
2: $A(j) \longleftarrow \phi\ forall$ node M_j
3: **for** $i = 1$ to MaxTask **do**
4: Let M_j be a node with maximum $t_{ij}; max_{1 \leq j \leq m}(t_{ij})$
5: Allocate task t_i to Node M_j
6: $A(j) \longleftarrow A(j) \cup \{t_i\}$
7: $T_j \longleftarrow T_j + t_{ij}$
8: **end for**
9: $T \longleftarrow max_j T_j$

A greedy algorithm that computes the *makespan* using a sorted or consistent ETC matrix is presented in Algorithm 5.2. The algorithm terminates exactly in $n = MaxTask$ steps.

Theorem 5.3.3. *sorted greedy resource allocation algorithm is a 3/2 approximation algorithm.*

Proof. Let machine M_k be the machine with the maximum load, on allocation of the first m tasks to the different machines in the HDCS. When we are to assign the next task t_{m+1}, we have already the information about the node, that has the maximum load. We have assumed that the task t_{i*} is allotted to the node M_k. If $i* \leq m$, then t_{i*} is the only task to be assigned to M_k. This is feasible because the first m tasks are assigned to the different nodes using the greedy resource allocation algorithm. Hence, the allocation algorithm is optimal as every node gets a single task. If $i* > m$, then by using Theorem 5.3.1, we can have

$$T_k \leq t_{i*k} + \frac{1}{m} \sum_{1 \leq j \leq m} L_j \tag{5.5}$$

where L_j is the total load on node M_j and $\frac{1}{m} \sum_{1 \leq j \leq m} L_j$ is the average load of the system. Using Equation 5.2, we have

$$\frac{1}{m} \sum_{1 \leq j \leq m} L_j \leq max(max_{1 \leq j \leq m} L_j \, , \frac{1}{m} \sum_{1 \leq j \leq m} L_j) \leq L_{max} \tag{5.6}$$

or,

$$\frac{1}{m} \sum_{1 \leq j \leq m} L_j \leq L_{max} \tag{5.7}$$

The tasks are appearing in the $SETC$ matrix in the order of $t_{1j} \geq t_{2j} \geq \cdots \geq t_{nj}$ for node M_j. If $i* > m$, for any arbitrary computing node M_j, we have $t_{i*j} \leq t_{(m+1)j} \leq t_{mj}$. Then by using Lemma 5.3.2 we have

$$t_{i*j} \leq (t_{mj} + t_{(m+1)j})/2 \leq L_{max}/2 \tag{5.8}$$

Using Equations 5.7 and 5.8 on Equation 5.5 we have:

$$T_k \leq L_{max}/2 + L_{max}$$

or,

$$T_k \leq (3/2)L_{max}$$

Hence the total load on node M_k is at most $(3/2)L_{max}$ □

Greedy algorithms are natural choices for heuristics and support the design of very fast approximation algorithms. We are able to established the *performance guarantee* of Algorithm 5.1 and 5.2 with approximation ratios of 2 and 3/2 respectively. So, Algorithm 5.2 is a polynomial time $3/2-$*approximation algorithm* for the minimization problem defined in 5.1 and always returns a solution whose value is at most 1.5 times of optimal value.

5.4 Conclusion

Approximation algorithms finds reasonably good solutions in run time bounds. This chapter presented $\rho -$ *approximation algorithm* to solve load balancing problems on the HDCSs with a central scheduler. The algorithms were proved to be absolute approximation algorithms with defined lower bound for n tasks on m machines. The approximation schemes are applied to the ETC matrix considering both task and machine heterogeneity. Both the algorithms are with polynomial time complexities and produces sub-optimal solutions for the *load balancing* problem.

Chapter 6

Decentralized Load Balancing Algorithm using Game theory

Decentralized resource allocation algorithms have been used to design polynomial time algorithms for intractable problems that provide solutions within the bounded proximity of the optimal solution. The load balancing problem in heterogeneous distributed system is modelled as a multi-player non-cooperative game with Nash equilibrium. The decision to allocate the resources in an HDCS are based upon the pricing model of computing resources using a bargaining game theory. In the process prior to executing a task, the heterogeneous computing nodes are participate in a non-cooperative game to reach an equilibrium. The non-cooperative framework adopted to allocate tasks by m servers is modelled as an $m-player$ game. We have evaluated the performance of two existing price-based job allocation schemes, namely the Global Optimal Scheme with Pricing (GOSP) and Nash Scheme with Pricing (NASHP). A modified version of each of theses schemes has been introduced to analyze the performance by considering the effect of pricing on system utilization.

6.1 Introduction

Distributed systems provides support for powerful computing infrastructures to solve computationally demanding problems. These computing resources are spread over the globe with different independent administrative domains, where the ownership of computing nodes are with individuals or organizations. Grids and clouds are such systems with

highly scalable computing infrastructures, which are regularly increasing with an increase in the user base. The ownership from distinct individuals or organizations requires a decentralized resource management system. Problems with high computing requirements are most suitable to use a distributed computing infrastructure [171].

In the decentralized approach of load balancing, all of the computing nodes in the HDCS are involved in taking the load balancing decision. The load balancing decisions are based on the dynamic state of information of the whole system. Game theory has been used by the researchers to model a situation where at least two interactive decision makers with diverging interests are involved [172]. Interactive decision making involves players (nodes) as the decision makers. In particular, the node M_j has a goal to minimize its load A_j by transferring a part of the load to the other computing nodes in the HDCS. This load transfer process may lead to a situation where loads on the nodes are balanced. This results in equilibrium when the node has any incentive to transfer its load to the other nodes. The state at which load exchange between any two of the computing nodes stops and tasks are executed without interruption at arriving nodes is termed as the *Nash equilibrium state*[171].

Classical game theory is a normative theory, in the sense that it expects players or agents to be perfectly rational and behave accordingly. In classical game theory, interactions between rational agents are modeled as games of two or more players that can choose from a set of strategies and the corresponding preferences [173]. The game theory assumes that players will compute *Nash equilibrium* and choose to play one such strategy. Application of algorithmic game theory can be found in [174], which lists several applications including networking and artificial intelligence along with basics of game theory. Resource allocation problems can be modelled as cooperative or non-cooperative games in heterogeneous environments as suggested in [172].

Cooperative game theory offers formal models to provide axiomatic solutions through a situation where an enforceable binding agreements between each pair of decision makers (players) are possible[175]. Game theoretic algorithms are designed to achieve convergence to the *Nash equilibrium* by modelling the related information. They assume that the players can observe the actions of the other players. Moreover, the decision makers have complete freedom of preplay communication to make joint agreements about their operating points [76].

Non-cooperative game theory is applicable to the interactive decision making process where the decision makers act individually without involving others in the negotiation process [176]. Dynamic load balancing on an HDCS can be formulated as a non-cooperative game among the m computing nodes (players) who act as the decision maker to distribute

the tasks among them. The non-cooperative or strategic game theory is being used to model the interactive decision making process by the multiple players. The process of independent decision making by schedulers with computing node leads to an equilibrium. The two different equilibrium condition possible are the *Wardrop equilibrium* and *Nash equilibrium* [76]. For an infinite number of tasks the Wardrop equilibrium results, when a task t_i cannot receive any further benefit by changing the allocation decision by the local scheduler. If the task t_i is allocated to node M_j by the node M_i then the earliest completion time of task t_i on M_j denoted as c_{ij}, such that $C_{ij} = min\{C_{ik}\}$; for $k = 1, \cdots, m$. When we have a finite number of tasks to be executed on an HDCS, the *Wardrop equilibrium* reduces to *Nash equilibrium*.

The dynamic load balancing problem is formulated as a non-cooperative game among computing nodes when tasks arrive with them for execution. The non-cooperative approach in this chapter considers computing nodes as decision makers. However, the nodes that are not allowed to cooperate during the decision making process. Each of the computing nodes optimizes locally to minimize the response time for the task arriving at the node. In this chapter dynamic load balancing problem has been investigated as a non-cooperative game, also established the Nash equilibrium under general assumption on the cost. We have proposed two dynamic load balancing schemes namely Global Optimal Scheme with Pricing Binary (GOSP_binary) and Nash Scheme with Pricing Binary (NASHP_binary) and evaluated their performance through simulation.

6.2 Related work

In an HDCS, the computational resources are distributed and used for variety of applications having different resource requirements. These requirements are from the users, who are likely to behave in a selfish manner and their behavior cannot be characterized using conventional techniques [177]. The idea of using game theory in load balancing is not completely novel. Game theory decision making models are used in two different forms i.e., as *cooperative games* and as *non-cooperative games* among the tasks or users in distributed computing systems. Grosu *et al.*[178] formulated the static load balancing problem in single class job distributed systems as a cooperative game among computers and presentd the structure of the *bargaining solution* that provides a *Pareto optimal allocation* to all of the jobs. Evendar *et al.* [179] studied the load balancing problem in unrelated parallel machines as a generalized ordinal potential game. A truthful mechanism for solving the static load balancing problem in distributed systems is addressed in [76, 177]. The effectiveness of the truthful mechanism had been ascertained on a hetero-

geneous system consisting of 16 computers with four different processing rates. A game theoretic pricing strategy for efficient job allocation in mobile grids has been presented in [180] to obtain the Nash bargaining solution. A cooperative game based capacity allocation scheme for utility computing environments can be found in [181], where the system converges to a unique equilibrium point if the users iteratively makes *best response* updates to their bids.

John Nash [182] has proposed a theory on *non-cooperative* games by contradicting the *n-person* game model of *Von Neumann* and *Morgenstern*. The contradiction is based on the absence of coalitions between each pair of participants. They acts independently, without collaboration or communication with each other. The notion of an equilibrium point is the basic ingredient of the theory proposed by Nash. It is established in [182] that a finite non-cooperative game always has at least one equilibrium point. A "dynamical" approach to the study of cooperative games based upon reduction to *non-cooperative* forms was discussed in [183] by Dechert. Further, the non-cooperative dynamic game was presented as a control problem. The Nash equilibrium solution found by this method is not only an open loop solution but it is also a feedback solution.

Most of the algorithms designed for decentralized load balancing in HDCSs were converged to *Nash equilibrium*. A *non-cooperative* game theoretic framework has been formulated by Grosu and Chronopoulos [184] for static load balancing problems in heterogeneous distributed systems. The concept of Nash equilibrium is used to design a new distributed load balancing algorithm. Penmatsa and Chronopoulos [185] studied a cooperative game theoretic model to solve a static load balancing problem for heterogeneous distributed systems. The cooperative game model is used with the solution based on the Nash bargaining model to provide Pareto optimality. Altman *et al.* [186] have investigated optimal load balancing strategies for a multi-class multi-server processor-sharing system with heterogeneous service capacities. The two different approaches namely, the centralized setting; and the decentralized, distributed non-cooperative setting have been presented to minimize its *weighted mean sojourn time* in the system. Paul *et al.* [187] introduced a non-model based approach for locally stable convergence to Nash equilibrium for static *non-cooperative* games with N players. In this non-model based approach players determine their actions by using only their own measured pay-off values. The players attain their Nash equilibrium without the need of model information by utilizing deterministic extremum seeking with sinusoidal perturbations. Khan and Ahmad [188] experimentally evaluated three game theocratic resource allocation mechanisms on a grid computing system by considering task and machine heterogeneity.

Most of the decentralized approaches use the partial information available with the

individual computing nodes to make suboptimal decisions [172]. The scope of applying game theoretic techniques to load balancing in distributed computer systems has been analysed in the context of Nash equilibrium by a majority of the researchers. To facilitate a game theoretic approach, the HDCS is viewed as the collection of computing resources that are under the supervision of the server with each node. The *scheduler* or *load balancer* is available as a component of the server to facilitate the task allocation. This chapter presents a non-cooperative game theoretic framework for dynamic load balancing in heterogeneous distributed systems with the goal of achieving Nash equilibrium.

6.3 Distributed system model

We consider a distributed computing system with distributed dynamic load balancing based on a heterogeneous distributed system model that consists of a set of m computing nodes (resources), connected by a communication network as shown in Figure 6.1. We have assumed that the jobs are atomic and can not be further subdivided, hence each is treated as a single task. Let there be a set of n independent tasks each with the expected time to compute on m machines represented by the ETC matrix given in Table 2.1. These tasks are to be assigned to any computing node $M_j \in M$. An assignment $A : T \longrightarrow M$ implies that every task t_i is to be assigned to a machine M_j with an expected time to compute t_{ij}. A task arriving at the node M_j may be executed at node M_j or transferred to another node M_l through the communication network by message passing. The task transformed from node M_j to node M_l receives its services at node M_l, no further migration for that task is permitted. The following assumptions are used to define decentralized dynamic load balancing in distributed systems:

- Tasks arrive in a single queue at each of the computing nodes.
- Task migrate within the HDCS takes place without any centralised control and only each of the tasks has a local view of the system on which it resides.
- All of the tasks know how many resources (machines) are available.
- The task transferred from the node M_i to node M_j is served at node M_j and is not transferred to any other node for execution.
- Each assigned task to the node resides in a queue that is processed in an FCFS order.

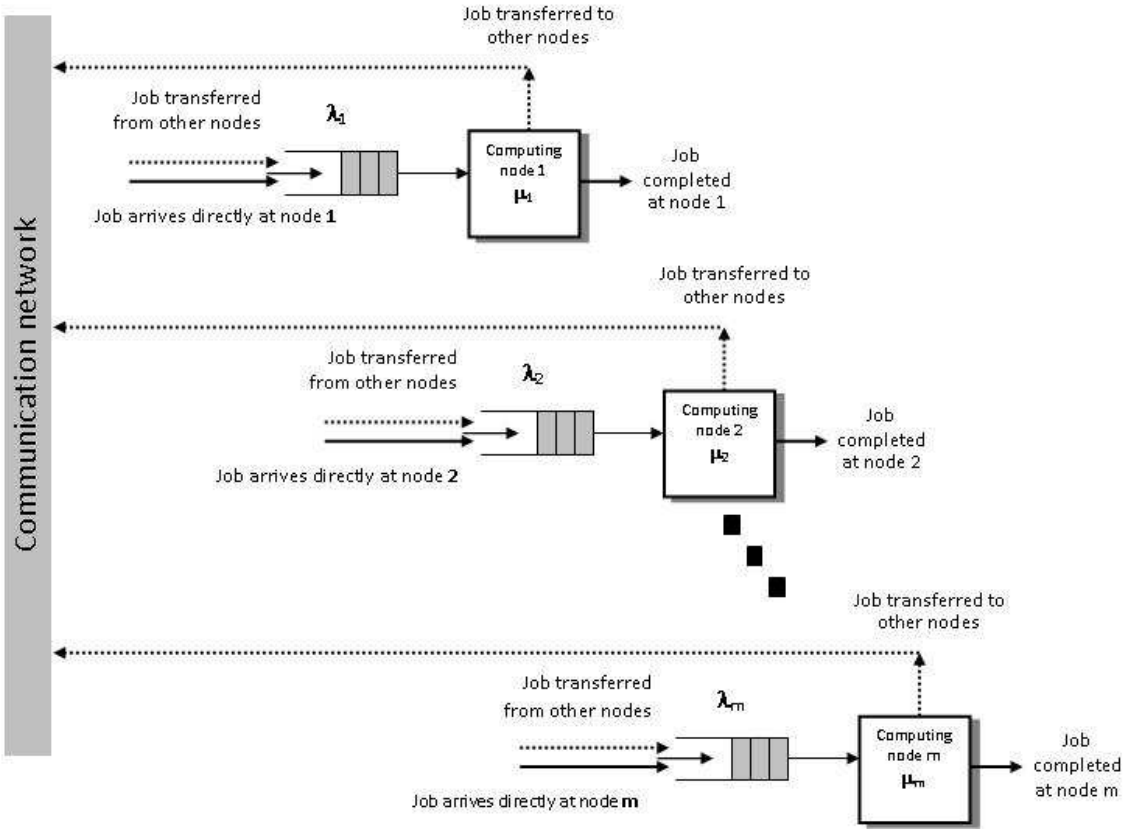

Figure 6.1: Decentralized Heterogeneous Distributed Computing System nodes

We model an HDCS with m heterogeneous computing nodes. Each computing node M_j has a scheduler known as the local scheduler, denoted as s_j. The local scheduler is responsible for load balancing decisions. Let Q_j be the queue associated with computing node M_j as shown in Figure 6.2. The local scheduler receives the tasks from the users through the queue and assigns them to the nodes of the HDCS or executes them locally. Processor(s) with the computing node is/are the principal computing element(s) of the node that executes and process the task assigned to it. We have assumed that all computational tasks can be executed on any computing nodes of the HDCS.

A task arriving at node M_j may be processed at node M_j or the scheduler s_j transfers the task to another node M_l through the communication network. Selection of the destination node is the decision of the scheduler s_j based upon the present state of the information on the HDCS. Let tasks arrive at node M_j following Poisson distribution with a mean arrival rate of λ_j. Expected times to compute the task t_i are represented as ETC matrix. The total task arrival rate to the system is denoted as λ, and defined as $\lambda = \sum_{i=1}^{m} \lambda_i$. The computational power of each computing node is denoted as μ_j (tasks

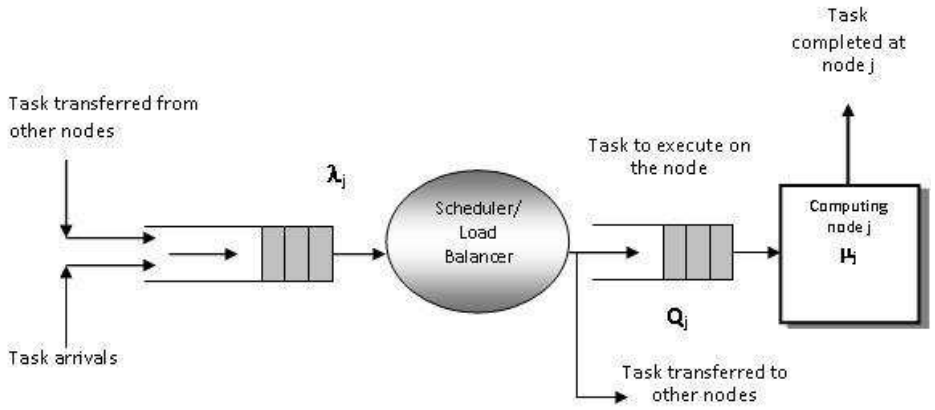

Figure 6.2: Node of decentralized Heterogeneous Distributed Computing System

per second). For stability, it is assumed that

$$\sum_{i=1}^{m} \lambda_i \; < \; \sum_{j=1}^{m} \mu_j$$

Each scheduler s_j keeps track of the price per unit of resource available for computing in a node M_j, denoted as p_{ji}. A separate price vectors are maintain by schedulers s_j on behalf of the task with node M_i. So we have a price vector corresponding to each task on m computing node. Let $r_{ij} \; > \; 0$ be the fraction of task with Q_i, that are migrated to other nodes for execution. Hence $\sum_{j=1}^{m} r_{ij} \; = \; 1$. Moreover the task assign to node M_j must not exceed the rate at which M_j operates. So for stability it should satisfy the following:

$$\sum_{i=1}^{m} r_{ij}\lambda_i \; < \; \mu_j$$

Let R_j be the vector that represents load balancing strategy for scheduler s_j and defined as $R_j \; = \; (r_{j1}, r_{j2}, \cdots, r_{jm})$. We can define $R \; = \; (R_1, R_2, \cdots, R_m)$ as the load allocation vector for HDCS, also known as strategy profile of load balancing game [184].

We consider a heterogeneous distributed system with m computing nodes(or computers) connected through a communication network as shown in Figure 6.3. Each computing node with communication link is modelled as $M/M/1$ queuing system. We have the similar assumptions on computing nodes as described in Section 2.3. However in the context of decentralized load balancing there is *no* interaction between *global scheduler* with the scheduler of other nodes in order to perform load distribution in HDCS.

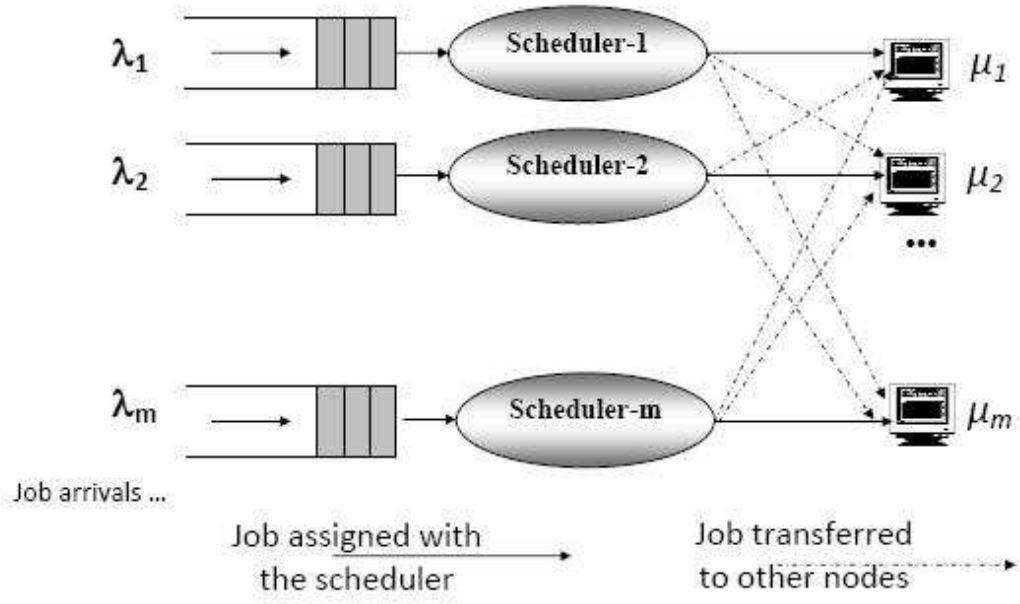

Figure 6.3: M/M/n queueing model of decentralized Heterogeneous Distributed Computing System

Let Q_j is the queue associated with the node M_j and represents number of task waiting to be served. If all the nodes are busy in executing task with them, then the total task n with the system is given as

$$n = \sum_{j=1}^{m} Q_j \; + \; m \tag{6.1}$$

Hence the expected response time at computing node M_j is given by

$$T_j(R) = \frac{1}{\mu_j - \sum_{k=1}^{m} r_{kj}\lambda_k} \tag{6.2}$$

The overall expected response time of scheduler s_j on node M_j in the system denoted as $D_j(R)$, and defined as

$$D_j(R) \; = \; \sum_{i=1}^{m} r_{ji} T_i(R) \; = \; \sum_{i=1}^{m} \frac{r_{ji}}{\mu_i - \sum_{k=1}^{m} r_{ki}\lambda_k} \tag{6.3}$$

Hence load balancing in this context is to find a feasible load balancing R_j by the scheduler S_j such that $D_j(R)$ will be minimized. The load balancing is *dynamic* because,

the decision by scheduler s_j depends on the load balancing information of other schedulers as $D_j(R)$ is the function of R.

Let $\psi_j(R)$ be the expected cost of computing node M_j, and can be computed as follows:

$$\psi_j(R) \;=\; \sum_{i=1}^{m} k_i p_{ji} r_{ji} T_i(R) \;=\; \sum_{i=1}^{m} \frac{k_i p_{ji} r_{ji}}{\mu_i - \sum_{k=1}^{m} r_{ki}\lambda_k}, \tag{6.4}$$

where k_i is assumed to be constant that maps the amount of computing resources at node M_i for a task t_j. Again p_{ji} be the agreed price obtained as a result of bargaining game between scheduler s_j and computing node M_i. The expected cost of the HDCS with m computing nodes can be given as:

$$\psi(R) \;=\; \frac{1}{\lambda}\sum_{j=1}^{m} \lambda_j \psi_j(R) \tag{6.5}$$

6.4 Load balancing problem as a dynamic game

Strategic decision making is often based on conceptual and quantitative model of the problem domain. Game theory is becoming more important and widely used as a tool to select quantitative strategies [189]. The tasks arrive to the system through the independent Poisson process are stored in the queue of different computing nodes. The non-cooperative load balancing game between the schedulers in a HDCS can be defined with three attributes [184]:

- *Players:* The scheduler with m computing node of HDCS.
- *Strategies:* Each scheduler s_j formulates a feasible load balancing strategy R_j.
- *Preference:* Each scheduler s_j preferences are defined as expected response time of scheduler D_j. Each player s_j prefers the strategy profile R over the strategy profile R' if and only if $D_j(R) \;<\; D_j(R')$

A solution to the above non-cooperative game is the optimal strategy for dynamic load balancing problem.

6.4.1 Nash equilibrium

In our model a computing node M_j tries to minimize its load L_j by sending part of the load to the other computing nodes in HDCS. To define load balancing an assumption is that the difference of the load of any two arbitrary node is to satisfy $|L_j - L_i| < \theta$, where θ is a small positive constant and less than the expected time to compute the

smallest task on fastest node in HDCS. In the context of load balancing, it is the state in which all nodes are satisfied. Hence the load exchange among the computing nodes are stopped when they are satisfied and the load can process without interruption from load arriving or finishing execution at a node. When all nodes are satisfied the system said to be in *Nash equilibrium* [171]. Otherwise a *Nash equilibrium* for this resource allocation m-player game is a sequence of action that generates allocation list or assignment Aj for each computing node M_j. Let A be an assignment or allocation of n task to m computing node, defined as $A = \{A(1), A(2), \cdots, A(m)\}$, where $A(j)$ is the set of task allocated to node M_j. Every task t_i may be allocated to a unique node. Load L_j on a node M_j, also corresponds to the assignment $A(j)$, such that $L_j = \sum_{t_i \in A(j)} t_{ij}$. Hence the response time of node M_j is the function of total load L_j and denoted as T_j as defined in Section 2.6. With the view the scheduler as independent selfish player seeking to minimize response time of their tasks, leads to design a game-theoretic model for load balancing problem.

A feasible assignment A is a Nash equilibrium if and only if for all task t_i allocated to node M_j is in A, and for all $M_k \in M$

$$T_j \leq T_k + t_{ik}$$

Using load balancing strategy R_j the *Nash equilibrium* for load balancing problem [184] can be defined.

Definition 6.1 (Nash equilibrium)**.** *A Nash equilibrium in load balancing game among m schedulers can be defined as the strategy R such that*

$$R_j \in \underset{R_j}{minimum}\ D_j(R); for R_j = (r_{j1}, r_{j2}, \cdots, r_{jm})$$

The decentralize dynamic load balancing problem on HDCS using a non-cooperative approach can be presented as an minimization problem to minimize $D_j(R)$ for all computing node M_j with a feasible load balancing strategy R_j, for all $j = 1, ..., m$. Hence the linear programming problem for load balancing can be stated as

$$\underset{R_j}{\text{minimize}}\ D_j(R) \tag{6.6}$$

subjected to:

$$r_{ji} \geq 0, \quad for\ i = 1, \ldots, m. \tag{6.7}$$

$$\sum_{i=1}^{m} r_{ji} = 1, \tag{6.8}$$

$$\sum_{k=1}^{m} r_{kj}\lambda_k < \mu_j, \quad for\ i = 1, \ldots, m. \tag{6.9}$$

Considering price agreed between scheduler and computing nodes, load balancing problem can be formulated as

$$\underset{R_j}{\text{minimize}}\ \psi_j(R) \tag{6.10}$$

subjected to:

$$r_{ji} \geq 0, \quad for\ i = 1, \ldots, m. \tag{6.11}$$

$$\sum_{i=1}^{m} r_{ji} = 1, \tag{6.12}$$

$$\sum_{k=1}^{m} r_{kj}\lambda_k < \mu_j, \quad for\ i = 1, \ldots, m. \tag{6.13}$$

6.4.2 Computation of optimal load fraction

The *load fraction* R of the system can be computed by calculating $R_i = (r_{i1}, r_{i2}, \cdots, r_{im})$ for each computing node M_i. The load fraction of each computing node M_i can be calculated by considering the amount of load assigned to M_i by other computing nodes in HDCS. Let μ_i is the processing rate of the node M_i, then *available processing rate* of computing node M_i as seen by scheduler s_j is defined as

$$\mu_i^j = \mu_i - \sum_{k=1,k\neq 1}^{m} r_{ki}\lambda_k$$

The tasks arrives independently to m computing nodes. It is assumed that estimated time of computation of each tasks are know in advance and can be represented by ETC matrix as discussed in Section 2.5. The *system utilization* denoted as ρ is computed as the ratio of total task arrival rate λ to the aggregate processing rate of the computing nodes in HDCS and defined as

$$\rho = \frac{\lambda}{\sum_{i=1}^{m} \mu_i} \tag{6.14}$$

The *system utilization* or *system utility* is assumed to be fixed form 0.2 to 0.9 to carry out the simulation. Algorithm 6.1 calculate the optimal load fraction for scheduler s_j running on M_j and has been adapted from [190].

6.4.3 Modified decentralized non-cooperative global optimal scheme

The optimal load fractions for all the computing nodes of HDCS can be computed with some communication between the schedulers of each computing node. In distributed com-

Algorithm 6.1 Best_fractions

Require: *Available processing rate* : $\mu_1^j, \mu_2^j, \cdots, \mu_m^j$,*Total arrival rate* : λ_j,
Price perunit vecotor : $p_{j1}, p_{j2}, \cdots, p_{jm}$, *The constant vector* : $k_1, k_2, \cdots, k_m$
Ensure: *Load fractions* : $r_{j1}, r_{j2}, \cdots, r_{jm}$

1: Sort the computing nodes in decreasing order of $\left(\frac{\mu_1^j}{\sqrt{\mu_1 k_1 p_{j1}}} \geq, \frac{\mu_2^j}{\sqrt{\mu_2 k_2 p_{j2}}} \geq \cdots \geq \frac{\mu_m^j}{\sqrt{\mu_m k_m p_{jm}}}\right)$
2: $t \longleftarrow \frac{\sum_{i=1}^{m} \mu_i^j - \lambda_j}{\sum_{i=1}^{m} \sqrt{\mu_i k_i p_{ji}}}$
3: **while** $t \geq \frac{\mu_m^j}{\sqrt{\mu_m k_m p_{jm}}}$ **do**
4: $r_{jm} \longleftarrow 0$
5: $m \longleftarrow m - 1$
6: $t \longleftarrow \frac{\sum_{i=1}^{m} \mu_i^j - \lambda_j}{\sum_{i=1}^{m} \sqrt{\mu_i k_i p_{ji}}}$
7: **end while**
8: **for** $i = 1, \cdots, m$ **do**
9: $r_{jm} \longleftarrow \frac{1}{\lambda_j}\left(\mu_i^j - t\sqrt{\mu_i k_i p_{ji}}\right)$
10: **end for**

puting system each scheduler s_j are the process running on respective computing node M_j. The process execution and message transfer are assumed to be asynchronous. It is also assumed that the message transmission delay is finite and unpredictable. Scheduler process consists of a sequential execution of its actions. These actions are atomic and the actions of a scheduler process are modeled as three types of events, namely, *internal events*, *message send events*, and *message receive events* [191]. A typical job allocation schemes executed by the scheduler is also consists of these three different events. For a message *msg*, let ***Send****(msg)* and ***Recv****(msg)* denote its send and receive events, respectively.

The proposed *GOSP_Binary* job allocation Algorithm 6.2 is an iterative algorithm adopted from original GOSP algorithm of Penmatsa and Chronopoulos [190] but uses the algorithm Best_Fractions_Binary given in Algorithm 6.2. The Table 6.1 lists the notations that are used in Algorithm 6.3.

The time complexity of the Algorithm 6.2 can be computed as follows:

- The contribution of the sorting process in Step 1 is $\bigcirc(m\ log\ m)$.
- The domain of the while loop from Step 4 to 16 is of $\bigcirc(m\ log\ m)$.

Algorithm 6.2 Best_Fractions_Binary

Require: *Available processing rate* : $\mu_1^j, \mu_2^j, \cdots, \mu_m^j$,*Total arrival rate* : λ_j,
Price perunit vecotor : $p_{j1}, p_{j2}, \cdots, p_{jm}$, *The constant vector* : $k_1, k_2, \cdots, k_m$
Ensure: *Load fractions* : $r_{j1}, r_{j2}, \cdots, r_{jm}$

1: Sort the computing nodes in decreasing order of
$\left(\frac{\mu_1^j}{\sqrt{\mu_1 k_1 p_{j1}}} \geq, \frac{\mu_2^j}{\sqrt{\mu_2 k_2 p_{j2}}} \geq \cdots \geq \frac{\mu_m^j}{\sqrt{\mu_m k_m p_{jm}}} \right)$
2: $t \longleftarrow \frac{\sum_{i=1}^{m} \mu_i^j - \lambda_j}{\sum_{i=1}^{m} \sqrt{\mu_i k_i p_{ji}}}$
3: $low \longleftarrow 1, high \longleftarrow m, temp = 1$
4: **while** $low \leq high$ **do**
5: $temp = \frac{(low + high)}{2}$
6: **if** $t \geq \frac{\mu_m^j}{\sqrt{\mu_m k_m p_{jm}}}$ **then**
7: $high \longleftarrow temp - 1$
8: **else**
9: $low \longleftarrow temp - 1$
10: **end if**
11: **for** $i = temp, \cdots, m$ **do**
12: $r_{jm} \longleftarrow 0$
13: $m \longleftarrow temp$
14: $t \longleftarrow \frac{\sum_{i=1}^{m} \mu_i^j - \lambda_j}{\sum_{i=1}^{m} \sqrt{\mu_i k_i p_{ji}}}$
15: **end for**
16: **end while**
17: **for** $i = 1, \cdots, m$ **do**
18: $r_{jm} \longleftarrow \frac{1}{\lambda_j} \left(\mu_i^j - t \sqrt{\mu_i k_i p_{ji}} \right)$
19: **end for**

- The for loop in the Step 17 contributes $\bigcirc(m)$.

Therefore the time complexity of the Algorithm 6.2 is $\bigcirc(m \; log \; m)$.

Every scheduler s_j executes global job allocation routine using Algorithm 6.3 independently on respective computing node. Once the load fraction is computed on a node, the computing nodes uses R_j to take allocation decision for the tasks with queue Q_j. This algorithm can be used periodically or when the system parameters are changed to recalculate R_j. The objective of *GOSP_Binary* algorithm is to minimize the expected cost of all the tasks that are executed in HDCS. The experiments are conducted with

Table 6.1: GOSP_Binary algorithm parameters

Notation	Description
j	Computing node number
l	Iteration number
R_j^l	Load fractions of node M_j at iteration number l
D_j^l	Expected execution time of node M_j at iteration number l
ε	Tolerance limit
Send(j,(p, l, action))	Send message (p, l, action) to node M_j
Recv(j,(p, l, action))	Receive message (p, l, action) from node M_j
action	specific action the computing node has to perform
p	is a real number variable
norm	L^l norm at iteration l, defined as: $norm = \sum_{j=1}^{m} \lvert D_j^{(l-1)} - D_j^{(l)} \rvert$

three different cost model of computing nodes namely *random*, *ascending* and *descending* by arranging computing nodes according to their processing rate.

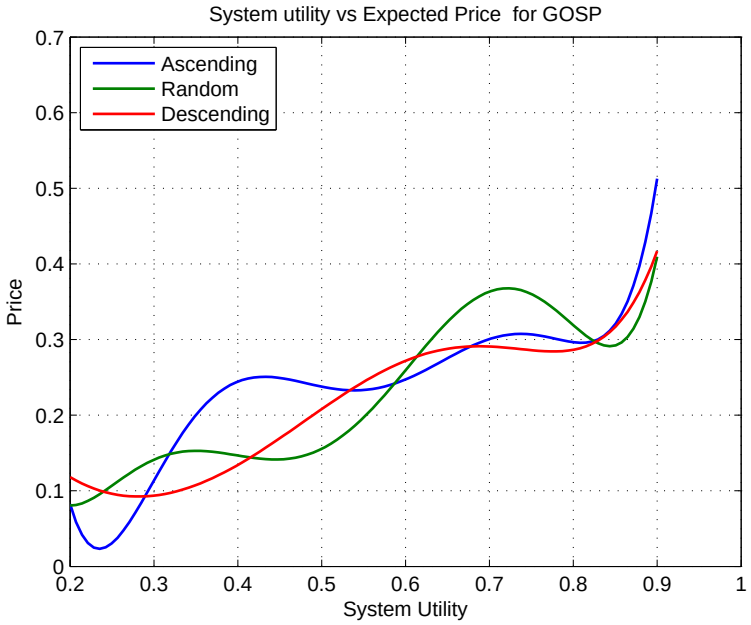

Figure 6.4: Expected price as function of system utilization based on GOSP

Algorithm 6.3 GOSP_Binary

Require: None. {*The routine to be executed by each server on respective node*}
Ensure: Updated load fraction.

1: Initialize: $R_j^{(0)} \longleftarrow 0; D_j^{(0)} \longleftarrow 0; l \longleftarrow 0; norm \longleftarrow 1; sum \longleftarrow 0;$
   tag $\longleftarrow CONTINUE; left \longleftarrow [(j-2)\ mod\ m]+1; right \longleftarrow [j\ mod\ m]+1$
2: **while** (1) **do**
3: **if** $(j = 1) : for\ computing\ node\ 1$ **then**
4: **if** $(l \neq 0)$ **then**
5: $Recv(left, (norm, l, tag))$
6: **if** $norm < \varepsilon$ **then**
7: $Send(right, (norm, l, STOP))$
8: **exit**
9: **end if**
10: $sum \longleftarrow 0$
11: $l \longleftarrow l + 1$
12: **end if**
13: **else**
14: $Recv(left, (sum, l, tag))$
15: **if** $tag = STOP$ **then**
16: **if** $(j \neq m)$ **then**
17: $Send(right, (sum, l, STOP))$
18: **exit**
19: **end if**
20: **end if**
21: **end if**
22: **for** $i = 1, \ldots, m$ **do**
23: $Compute\ \mu_i^j\ for\ each\ node: \ \mu_i^j \longleftarrow \mu_i - \sum_{k=1,k}^{m} r_{ki}\lambda_k$
24: **end for**
25: $R_j^{(l)} \longleftarrow$ **Best_Fractions_Binary**$(\mu_1^j, \mu_2^j, \cdots, \mu_m^j, \lambda_j, p_{j1}, p_{j2}, \cdots, p_{jm}, k_1, k_2, \cdots, k_m)$
26: $ComputeD_j^{(l)}$
27: $sum \longleftarrow sum + |D_j^{(l-1)} - D_j^{(l)}|$
28: **Send** $(right, (sum, l, CONTINUE))$
29: **end while**

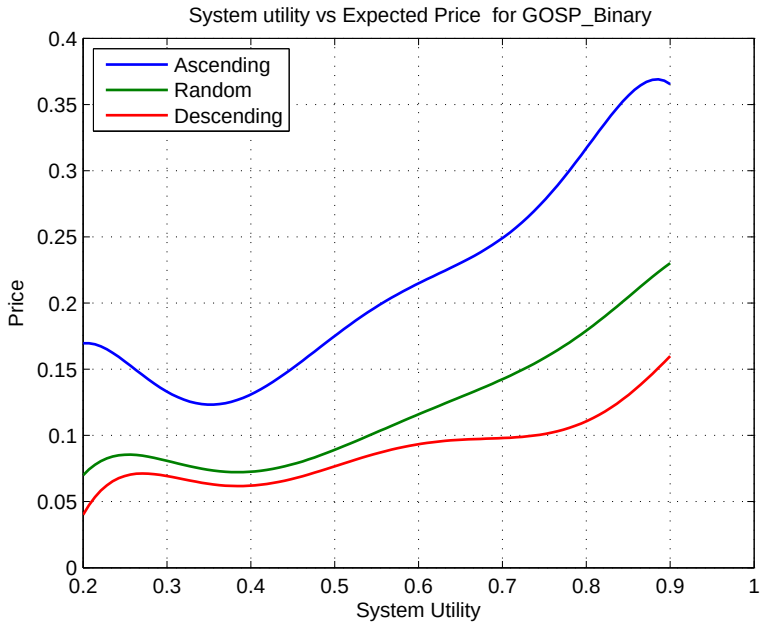

Figure 6.5: Expected price as function of system utilization based on GOSP_Binary

6.4.4 Modified decentralized non-cooperative Nash scheme

In general, a *Nash equilibrium* with different computing node, is a state in which no node has the incentive to change her current decision[192]. In particular the computing nodes have no incentive to reallocate their task to other nodes of the HDCS. Let L_i and L_j be the load with computing node M_i and M_j respectively. If $L_j < L_i$, then migrate task from M_i to M_j . We have modified the **Best Reply** algorithm given in [184] to design a new Algorithm 6.4 called ***Best_Reply_Binary*** . The complexity of the Algorithm 6.4 is $\bigcirc(m\ log\ m)$. The available processing rates can be calculated from the queuing theory models for a specific total arrival rate λ against a fixed system utility ρ.

The *Nash algorithm* for load balancing are designed to attain Nash equilibrium that provides a scheduler optimal operation point for the HDCS. The Nash scheme has been designed using Algorithm 6.4 and designated as **Best_Reply_Binary**. The Nash algorithm for resource allocation is called **NASHP_Binary** and listed as Algorithm 6.5. This is identical to the Algorithm 6.3, with the procedure **Best_Fractions_Binary** replaced by **Best_Reply_Binary** at step number 25 to compute R_j for every computing node M_j. The scheduler with the computing nodes are responsible for execution of the Nash scheme periodically. The task arrival rates at different computing node at fixed intervals of time is considered to facilitate discrete event simulation. The computation of Nash equilib-

Algorithm 6.4 Best_Reply_Binary

Require: *Available processing rate* : $\mu_1^j, \mu_2^j, \cdots, \mu_m^j$,*Total arrival rate* : λ_j
Ensure: *Load fractions* : $r_{j1}, r_{j2}, \cdots, r_{jm}$

Sort the computing nodes in decreasing order of their available processing rate $\left(\mu_1^j \geq, \mu_2^j \geq \cdots \geq \mu_m^j\right)$
$t \longleftarrow \dfrac{\sum_{i=1}^{m} \mu_i^j - \lambda_j}{\sum_{i=1}^{m} \sqrt{\mu_i^j}}$
$low \longleftarrow 1, high \longleftarrow m, temp = 1$
while $low \leq high$ **do**
  $temp = \dfrac{(low + high)}{2}$
  if $t \geq \sqrt{\mu_m^j}$ **then**
    $high \longleftarrow temp - 1$
  else
    $low \longleftarrow temp - 1$
  end if
  for $i = temp, \cdots, m$ **do**
    $r_{jm} \longleftarrow 0$
    $m \longleftarrow temp$
    $t \longleftarrow \dfrac{\sum_{i=1}^{m} \mu_i^j - \lambda_j}{\sum_{i=1}^{m} \sqrt{\mu_i^j}}$
  end for
end while
for $i = 1, \cdots, m$ **do**
  $r_{jm} \longleftarrow \dfrac{1}{\lambda_j}\left(\mu_i^j - t\sqrt{\mu_i^j}\right)$
end for

rium requires some communication between computing nodes. The algorithm operates in a round-robin fashion to obtain load balancing strategies for the nodes in HDCS.

6.5 Experiments and results

The system has been implemented in Matlab(R2008a) environment and simulation have been performed to analyse the non-cooperative load balancing through the computation of workload fractions for every computing node M_j. The HDCS system assumed for simulation are with 60 heterogeneous computing nodes. Let p_j be the price associated with node M_j obtained through bargain game by severs associated with each computing node. The relative processing rate of computing node M_j denoted as α_j and defined as the ratio of the processing rate of node μ_j to the minimum of $\mu_1, \mu_2, \ldots, \mu_m$. Let μ_{min} be

Algorithm 6.5 NASHP_Binary

Require: None. {*The routine to be executed by each server on respective node*}
Ensure: Updated load fraction.

Initialize: $R_j^{(0)} \longleftarrow 0; D_j^{(0)} \longleftarrow 0; l \longleftarrow 0; norm \longleftarrow 1; sum \longleftarrow 0;$
tag $\longleftarrow CONTINUE; left \longleftarrow [(j-2)\ mod\ m]+1; right \longleftarrow [j\ mod\ m]+1$
while (1) **do**
  if $(j = 1) : for\ computing\ node\ 1$ **then**
    if $(l \neq 0)$ **then**
      $Recv(left, (norm, l, tag))$
      if $norm < \varepsilon$ **then**
        $Send(right, (norm, l, STOP))$
        exit
      end if
      $sum \longleftarrow 0$
      $l \longleftarrow l + 1$
    end if
  else
    $Recv(left, (sum, l, tag))$
    if $tag = STOP$ **then**
      if $(j \neq m)$ **then**
        $Send(right, (sum, l, STOP))$
        exit
      end if
    end if
  end if
  for $i = 1, \ldots, m$ **do**
    $Compute\ \mu_i^j\ for\ each\ node: \ \mu_i^j \longleftarrow \mu_i - \sum_{k=1,k}^{m} r_{ki}\lambda_k$
  end for
  $R_j^{(l)} \longleftarrow$ **Best_Reply_Binary**$(\mu_1^j, \mu_2^j, \cdots, \mu_m^j, \lambda_j, p_{j1}, p_{j2}, \cdots, p_{jm}, k_1, k_2, \cdots, k_m)$
  $ComputeD_j^{(l)}$
  $sum \longleftarrow sum + |D_j^{(l-1)} - D_j^{(l)}|$
  Send $(right, (sum, l, CONTINUE))$
end while

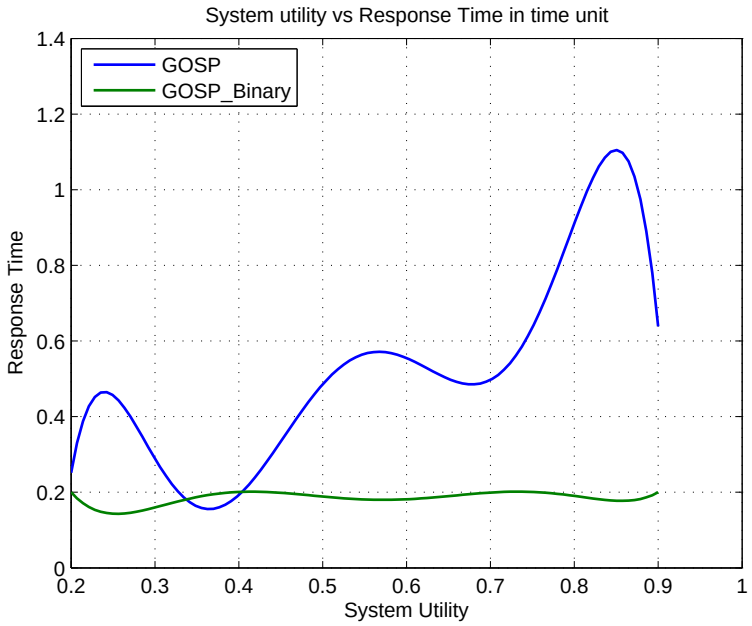

Figure 6.6: System utility vs expected response time on GOSP and GOSP_Binary

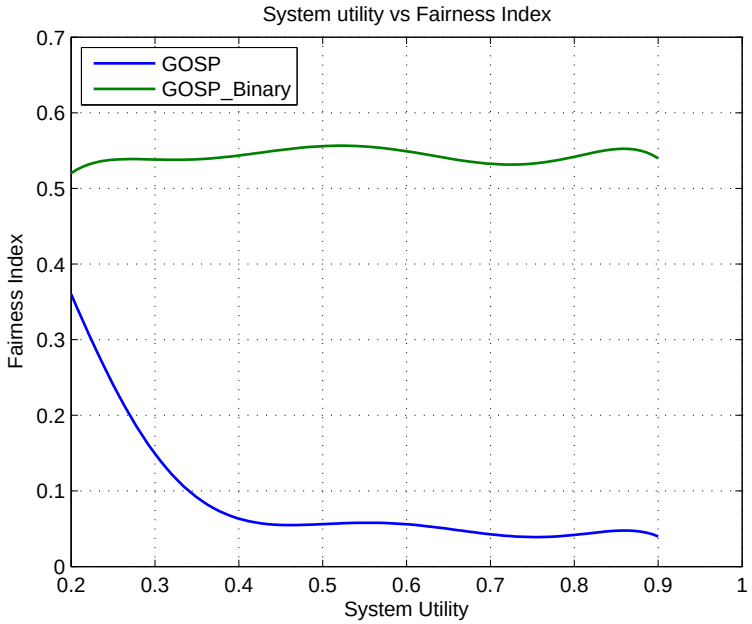

Figure 6.7: System utility vs fairness index on GOSP and GOSP_Binary

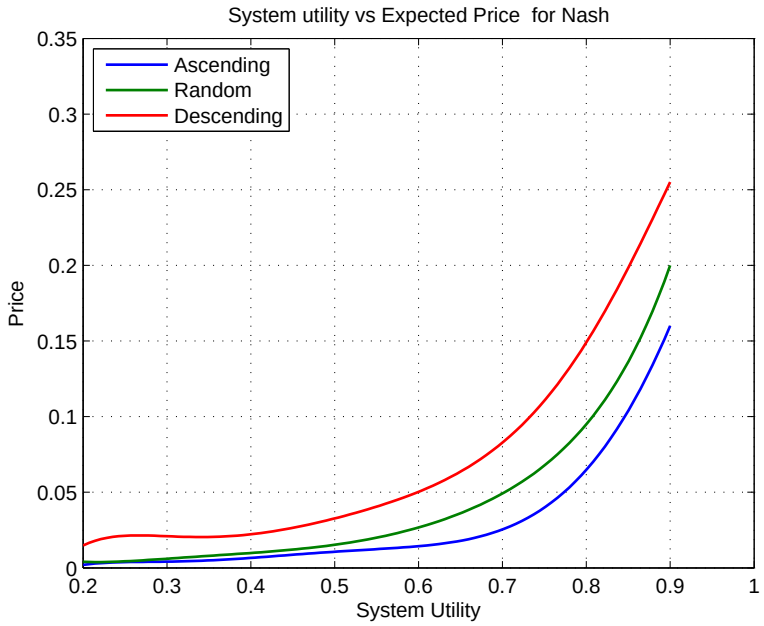

Figure 6.8: Expected price as a function of system utility on NASHP

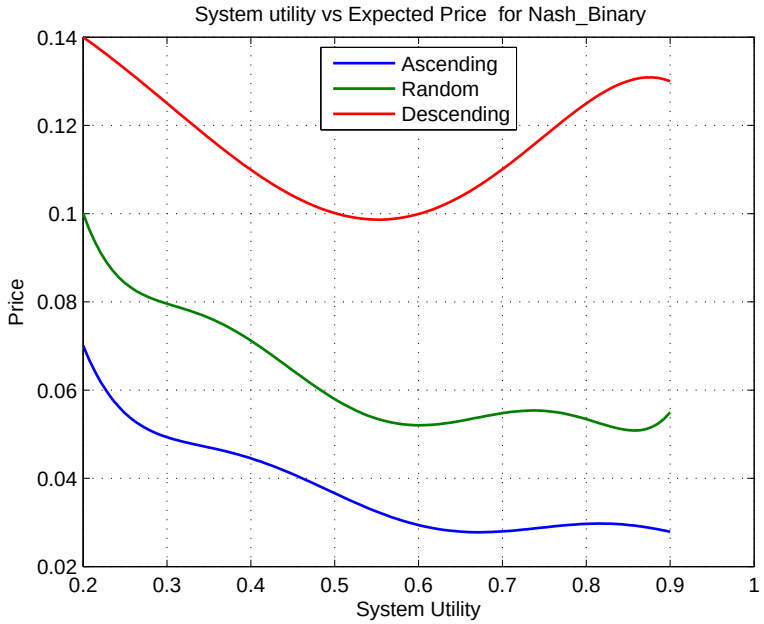

Figure 6.9: Expected price as a function of system utility on NASHP_Binary

the processing rate of the slowest computing node. Hence relative processing rate defined as $\frac{\mu_j}{\mu_{min}}$. For every experiment the total task arrival rate λ can be obtained from the Equation 6.14. We have conducted simulation experiment by varying *system utilization* ρ from 0.2 to 0.9 with the interval of 0.1.

Each computing nodes have been modelled as $M/M/1$ queueing system. Three different scenario for the experiments have been designed by considering the arrangement of the computing nodes in the order of their service rate μ_j. The different scenario of HDCS can be obtained by arranging computing nodes in *decreasing order*, *increasing order* and *random order* of their service rate. Then the ETC matrix corresponding to random order of the node are defined as inconsistent ETC matrix given in Section 2.5. The ordering of computing node represents the expected time to compute the task at consistent ETC matrix [24]. The pricing vector considered for simulation also follows the similar consideration for these three scenario. Figure 6.4 represents performance of GOSP algorithm [190] using average of 10 simulation. The result indicates total price charged by the system as a function of system utilization. We can have more load on computing system leads to the higher expected response time. Figure 6.5 indicates the performance of our proposed algorithm **GOSP_Binary**. The result obtained in Figure 6.5 are the average of 10 simulation run to study the performance of Algorithm 6.3.

To compare the performance of our proposed scheme, we have used two performance metric, *fairness index*, and *response time*. The *fairness index* of the HDCS with m heterogeneous computing nodes can be computed as

$$I(D) = \frac{[\sum_{j=1}^{m} D_j]^2}{m \sum_{j=1}^{m} D_j^2}, \tag{6.15}$$

where D_j is the expected execution time of node M_j and can be computed using Equation 6.3 and D is the total expected response time of the HDCS. Figure 6.6 shows the performance comparison between **GOSP** and **GOSP_Binary**. The proposed algorithm **GOSP_Binary** shows the better expected response time for the system. Figure 6.7 presents the fairness index for the HDCS with different value of system utilization raging from 0.2 to 0.9. The proposed algorithm **GOSP_Binary** shows remarkable improvement in fairness index.

The Nash schemes are designed to minimize the cost of individual computing nodes. Figure 6.8 shows the performance of NASHP algorithm [184] by varying system utilization raging from 0.2 to 0.9 for the three different scenario of computing nodes. The result obtained are the average of 10 simulation run. Figure 6.9 shows the performance of our proposed algorithm **NASHP_Binary** by varying system utilization, and found to be

more effective in terms of total cost of the system with three different price vector for higher system utilization. The expected price performance of the ascending price vector lies below the descending and random price vector corresponding to the processing rate of the computing nodes. Hence better performance can be obtained with arranging computing nodes in ascending order of processing rate.

6.6 Conclusion

Decentralize decision making by finite heterogeneous computing nodes have been modelled as non-cooperative game between the heterogeneous computing node. A non-cooperative decentralized game-theoretic framework has been presented for HDCS. The simulation excrements are based upon the mathematical model discussed in this chapter uses the performance metric *price*, *response time*, and *fairness index*. We have proposed two algorithm **GOSP_Binary** and **NASHP_Binary** to compute load fractions to allocate the tasks to the computing nodes of HDCS. It has been observed that, the algorithm **GOSP_Binary** further minimizes the cost of the entire computing system and so is advantageous when the system optimum is required. Hence, it is also fair to the schedulers and so as to the users. The scheme **NASHP_Binary** also further minimizes the cost for each computing nodes in the system by computing a feasible assignment that results load balancing strategy.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Load balancing problem in a Heterogeneous Distributed Computing System deals with allocation of tasks to the computing nodes, so that nodes are evenly loaded. In a heterogeneous distributed computing system the computational power of the computing entities are possibly different for each processor or node. The complexity of dynamic load balancing increases with the size of the HDCS and becomes difficult to solve effectively.

This thesis presents dynamic load allocation strategies for n independent tasks and m computing nodes in a heterogeneous distributed computing system through centralized or decentralized control. The load balancing strategies in HDCS aims to maintain a balanced execution of tasks while using the computational resources with computing node. The load balancing problem using *centralized approach* has been formulated considering task and machine heterogeneity as a linear programming problem to minimize the time by which all complete their execution. The load balancing strategies have been designed for dynamic load balancing with three different algorithm paradigms as (i) greedy algorithms,(ii) iterative heuristic algorithms,and (iii) approximation algorithm.

The system and task heterogeneity are modelled with expected time to compute(ETC) matrix. A batch mode heuristic has been used to design dynamic load balancing algorithm for heterogeneous distributed computing system with four different type of machine heterogeneity. A number of experiments has been conducted to study the performance of greedy load balancing algorithms with three different arrival rate for the task. A better performance of the algorithms are observed with higher degree of heterogeneity in HDCS with consistent and inconsistent task matrix.

A new codification scheme suitable to *simulated annealing* and *genetic algorithm* has been introduced to design dynamic load balancing algorithms for HDCS. These stochastic iterative load balancing algorithms uses sliding window techniques to select a batch of tasks, and allocates them to the computing nodes in HDCS. The effect of genetic algorithm and simulated annealing based dynamic load balancing scheme has been compared with load balancing strategies based on *first-fit* and *randomized* heuristic. The proposed dynamic genetic algorithm based load balancer has been found to be effective, especially in the case of a large number of tasks.

An analysis and design of two approximation algorithms for load balancing is presented with reference to ETC matrix for heterogeneous distributed computing systems with *makespan* as performance metric. The two proposed approximation schemes has been compared with an optimal solution computed as *lower bound* and proved to be *2-approximation* and *3/2 approximation* algorithm.

The decentralize load balancing problem in heterogeneous distributed system is modelled as multi player non-cooperative game with Nash equilibrium. In the process prior to execute a task, the heterogeneous computing nodes are participate in a non-cooperative game to reach an equilibrium. Two different types of decentralize load balancing problem has been presented in this thesis as minimization problems with *price*, *response time*, and *fairness index*. Two algorithms have been proposed to compute load fraction. These algorithms are used to design decentralized load balancing strategies to minimize the cost of the entire computing system leading to load balanced. It has been found that, the modified algorithms *GOSP_Binary* and *NASHP_Binary* further minimizes *the expected response time of scheduler* and *the cost for each computing nodes* respectively.

7.2 Limitations and Future work

The simulations studies with computers are subjected to some assumptions that leads to design of feasible simulation model to carry out experimentation. The simulation study of load balancing algorithms assumed that in a HDCS, if all the computing nodes are busy then a assigned task will keep on waiting in the waiting queue with central scheduler which is of infinite length. The task model used in this thesis are assumed the rate of arrival of the task to be Poisson distribution with arrival rate λ. However there are few instances where the task arrival follows different distributions. The task models used in this thesis are expressed as ETC matrix, where as researchers are also using task model as DAG. All the simulation experiments are assumed that the estimation of the execution time for each task on different computing nodes is know in advance and follows a uniform

distribution. The communication cost has not been featured is the system model of HDCS, hence this limits the scope of the effectiveness of load balancing algorithms for large HDCS.

As distributed systems continue to grow in scale, in heterogeneity, and in diverse networking technology, they are presenting challenges that need to be addressed to meet the increasing demands of better performance and services for various distributed application. For future research, the design of dynamic load balancing strategy can be sought for the HDCS that uses different class of task with specific resource requirements.

The performance degradation of HDCS are mostly due to the failure of one or more computing nodes. However many high-performance application requires high availability. Some high availability applications on distributed computing platform are military applications, 24×7 healthcare applications, international business applications etc. One of such application is multi-class applications that requires high availability on HDCS. In particular the *multi-class* application are consists of tasks of multiple classes that are characterized by *their distinctive arrival rates*, *execution time distributions*, and *availability requirements*. This becomes the basis of a new research domain to design dynamic resource allocation algorithms for high-performance applications with high availability.

Power management in distributed computing system is widely recognized to be an important research problem. The energy consumed by the distributed computing system can be conceptualized as the sum of energy consumed by the system components over time period while executing tasks until the complete execution of the tasks. Growing demand energy-efficient resource allocation in distributed computing needs the modelling of dynamic load balancing problem for HDCS with additional energy constraint for uncertain *task execution times* and *communication times*. Further study can be made on dynamic energy efficient resource allocation strategy by considering the architecture of computing nodes and power requirement by computing resources. Parallel I/O has been an active research area in High Performance Computing(HPC) for over two decades. Energy efficient and parallel I/O performance are two critical measures in HPC. Scope of further research can be carried out on energy efficient dynamic allocation of I/O intensive tasks on HPC.

A highly heterogeneous computing environment systems are becoming popular with the scalability of multi-core CPUs, graphics processing unit, distributed file systems that supported distributed computing. Some of the computing applications requires the task in the form of divisible load. This leads to the problem of resource allocation of *divisible load* and *load scheduling* in highly heterogeneous distributed system. Research is also required to study and design *inter* and *intra-node* load balancing using divisible load.

References

[1] K. Hwang, G.C. Fox, and JJ Dongarra. *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann, 2012. [cited at p. 1, 2, 18]

[2] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, simulations, and Advanced Topics*. Wiley Series on Parallel and Distributed Computing. John Wiley and Sons Inc., 2000. [cited at p. 1]

[3] T.V. Gopal, N.S. Nataraj, C. Ramamurthy, and V. Sankaranarayanan. Load balancing in heterogenous distributed systems. *Microelectronics Reliability*, 36(9):1279–1286, 1996. [cited at p. 1, 9]

[4] H.J. Siegel, H.G. Dietz, and J.K. Antonio. Software support for heterogeneous computing. *ACM Computing Surveys (CSUR)*, 28(1):237–239, 1996. [cited at p. 1]

[5] Jie Wu. *Distributed System Design*. CRC press, 1999. [cited at p. 1, 3, 5, 6, 19, 25, 101]

[6] A.Y. Zomaya and Y.H. Teh. Observations on using genetic algorithms for dynamic load-balancing. *Parallel and Distributed Systems, IEEE Transactions on*, 12(9):899–911, 2001. [cited at p. 1, 4, 5, 6, 8, 24, 25, 42, 69, 70, 75, 76, 85, 86, 89, 93, 99]

[7] M. Maheswaran and H.J. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh*, pages 57–69. IEEE, 1998. [cited at p. 2]

[8] M. Maheswaran, S. Ali, HJ Siegal, D. Hensgen, and R.F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 30–44. IEEE, 1999. [cited at p. 2, 30]

[9] M. Maheswaran, T.D. Braun, and H.J. Siegel. Heterogeneous distributed computing. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 1999. [cited at p. 2]

[10] Jean Dollimore George Coulouris and Tim Kindberg. *Distributed Operating System-Concepts and Design*. Addison Wesley, second edition, 2000. [cited at p. 2]

[11] Vijay K. Garg. *Elements of Distributed Computing*. Wiley-Interscience: John Wiley and Sons, Inc. Publication, 2006. [cited at p. 2]

[12] Sukumar Ghosh. *Distributed systems: an algorithmic approach*. CRC press, 2010. [cited at p. 2, 3]

[13] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Pearson Education, Inc., 2002. [cited at p. 2]

[14] Gamal Attiya and Yskandar Hamam. Task allocation for minimizing programs completion time in multicomputer systems. In *Computational Science and Its Applications–ICCSA 2004*, pages 97–106. Springer, 2004. [cited at p. 3]

[15] Jie Li and Hisao Kameda. Load balancing problems for multiclass jobs in distributed/parallel computer systems. *Computers, IEEE Transactions on*, 47(3):322–332, 1998. [cited at p. 3, 5]

[16] James Bunch, Jack Dongarra, Cleve Moler, and G.W. Stewart. Linpack users guide. *SIAM, Philadelphia, PA*, 1979. [cited at p. 3]

[17] R. Subrata, A.Y. Zomaya, and B. Landfeldt. Artificial life techniques for load balancing in computational grids. *Journal of Computer and System Sciences*, 73(8):1176–1190, 2007. [cited at p. 3, 4, 69, 70]

[18] T. Rotaru and H.H. Nägeli. Fast algorithms for fair dynamic load redistribution in heterogeneous environments. *Applied numerical mathematics*, 49(1):81–95, 2004. [cited at p. 3]

[19] B. Hamidzadeh, Y. Atif, and D.J. Lilja. Dynamic scheduling techniques for heterogeneous computing systems. *Concurrency: Practice and Experience*, 7(7):633–652, 1995. [cited at p. 3]

[20] Sukumar Ghosh. *Distributed systems: an algorithmic approach*, volume 13. Chapman Hall/CRC, 2006. [cited at p. 4]

[21] Y. Murata, T. Inaba, H. Takizawa, and H. Kobayashi. A distributed and cooperative load balancing mechanism for large-scale P2P systems. In *Applications and the Internet Workshops, 2006. SAINT Workshops 2006. International Symposium on*, pages 126–129. IEEE, 2006. [cited at p. 4]

[22] Anil Kumar Tripathi, Biplab Kumer Sarker, Naveen Kumar, and Deo Prakash Vidyarthi. A GA based multiple task allocation considering load. *International Journal of High Speed Computing*, 11(04):203–214, 2000. [cited at p. 4, 69, 70]

[23] W. Shen, Y. Li, H. Ghenniwa, C. Wang, et al. Adaptive negotiation for agent-based grid computing. *Journal of the American Statistical Association*, 97(457):210–214, 2002. [cited at p. 4]

[24] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, and Debra Hensgen. Task execution time modeling for heterogeneous computing systems. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pages 185–199. IEEE, 2000. [cited at p. 4, 22, 23, 33, 34, 46, 72, 129]

[25] Y. Azar, A. Epstein, and L. Epstein. Load balancing of temporary tasks in the lp norm. *Theoretical computer science*, 361(2):314–328, 2006. [cited at p. 4]

[26] M. Sriram Iyengar and M. Singhal. Effect of network latency on load sharing in distributed systems. *Journal of parallel and distributed Computing*, 66(6):839–853, 2006. [cited at p. 4]

[27] H.L. Chen, J.R. Marden, and A. Wierman. The effect of local scheduling in load balancing designs. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):110–112, 2008. [cited at p. 5]

[28] T.L. Casavant and J.G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *Software Engineering, IEEE Transactions on*, 14(2):141–154, 1988. [cited at p. 5, 7, 16]

[29] F. Spies. Modeling of optimal load balancing strategy using queueing theory. *Microprocessing and microprogramming*, 41(8):555–570, 1996. [cited at p. 5, 8, 36]

[30] A.M. Alakeel. A guide to dynamic load balancing in distributed computer systems. *International Journal of Computer Science and Network Security (IJCSNS)*, 10(6):153–160, 2010. [cited at p. 5]

[31] W. Becker and R. Pollak. Efficiency of server task queueing for dynamic load balancing. In *University of Stuttgart, Institute*, 1994. [cited at p. 5, 16]

[32] S. Muthukrishnan and R. Rajaraman. An adversarial model for distributed dynamic load balancing. In *Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures*, pages 47–54. ACM, 1998. [cited at p. 5]

[33] M.R. Garey and D.S. Johnson. *Computing and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979. [cited at p. 6, 11, 99]

[34] Wen-Chiung Lee, Chin-Chia Wu, and Peter Chen. A simulated annealing approach to makespan minimization on identical parallel machines. *The International Journal of Advanced Manufacturing Technology*, 31(3):328–334, 2006. [cited at p. 6, 69, 71]

[35] A.E. El-Abd and M.I. El-Bendary. A neural network approach for dynamic load balancing in homogeneous distributed systems. In *System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference on*, volume 1, pages 628–629. IEEE, 1997. [cited at p. 6]

[36] J. Watts and S. Taylor. A practical approach to dynamic load balancing. *Parallel and Distributed Systems, IEEE Transactions on*, 9(3):235–248, 1998. [cited at p. 6, 29]

[37] Z. Zeng and B. Veeravalli. Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks. *Computers, IEEE Transactions on*, 55(11):1410–1422, 2006. [cited at p. 6, 33]

[38] C.Z. Xu and F.C.M. Lau. Iterative dynamic load balancing in multicomputers. *Journal of the Operational Research Society*, pages 786–796, 1994. [cited at p. 7, 25]

[39] N.G. Shivaratri, P. Krueger, and M. Singhal. Load distributing for locally distributed systems. *Computer*, 25(12):33–44, 1992. [cited at p. 7, 9, 25, 27]

[40] Michael Boyer, Kevin Skadron, Shuai Che, and Nuwan Jayasena. Load balancing in a changing world: dealing with heterogeneity and performance variability. In *Conf. Computing Frontiers*, page 21, 2013. [cited at p. 7]

[41] Shaojun Zou. Analysis and algorithm of load balancing strategy of the web server cluster system. In *Communications and Information Processing*, pages 699–706. Springer, 2012. [cited at p. 7]

[42] Orly Kremien and Jeff Kramer. Methodical analysis of adaptive load sharing algorithms. *Parallel and Distributed Systems, IEEE Transactions on*, 3(6):747–760, 1992. [cited at p. 7]

[43] P.Y. Yin, S.S. Yu, P.P. Wang, and Y.T. Wang. Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization. *Journal of Systems and Software*, 80(5):724–735, 2007. [cited at p. 8]

[44] A. Aubry, A. Rossi, M.L. Espinouse, and M. Jacomino. Minimizing setup costs for parallel multi-purpose machines under load-balancing constraint. *European Journal of Operational Research*, 187(3):1115–1125, 2008. [cited at p. 8]

[45] G. Attiya and Y. Hamam. Task allocation for maximizing reliability of distributed systems: a simulated annealing approach. *Journal of Parallel and Distributed Computing*, 66(10):1259–1266, 2006. [cited at p. 8, 9, 71, 76, 81]

[46] E. Altman, U. Ayesta, and B.J. Prabhu. Load balancing in processor sharing systems. *Telecommunication Systems*, 47(1):35–48, 2011. [cited at p. 8]

[47] K. Li. Minimizing the probability of load imbalance in heterogeneous distributed computer systems. *Mathematical and computer modelling*, 36(9):1075–1084, 2002. [cited at p. 8]

[48] OJ Boxma, Ger Koole, and Zhen Liu. *Queueing-theoretic solution methods for models of parallel and distributed systems*. Centrum voor Wiskunde en Informatica, Department of Operations Research, Statistics, and System Theory, 1994. [cited at p. 8, 36]

[49] James Caffrey and Graham Hitchings. Makespan distributions in flow shop scheduling. *International Journal of Operations & Production Management*, 15(3):50–58, 1995. [cited at p. 8]

[50] R Rindzevičius, D Poškaitis, and B Dekeris. Performance measures analysis of m/m/m/k/n systems with finite customer population. *Electr. Electr. Eng*, 3(67):65–70, 2006. [cited at p. 8]

[51] Victor F. Nicola, Vidyadhar G. Kulkarni, and Kishor S. Trivedi. Queueing analysis of fault-tolerant computer systems. *Software Engineering, IEEE Transactions on*, (3):363–375, 1987. [cited at p. 8]

[52] K.Y. Kabalan, W.W. Smari, and J.Y. Hakimian. Adaptive load sharing in heterogeneous systems: Policies, modifications, and simulation. *International Journal of Simulation, Systems, Science and Technology*, 3(1-2):89–100, 2002. [cited at p. 8]

[53] Howard Jay Siegel and Shoukat Ali. Techniques for mapping tasks to machines in heterogeneous computing systems. *Journal of Systems Architecture*, 46(8):627–639, 2000. [cited at p. 8, 30, 31]

[54] Fangpeng Dong and Selim G Akl. Scheduling algorithms for grid computing: State of the art and open problems. *School of Computing, Queens University, Kingston, Ontario*, 2006. [cited at p. 8, 24]

[55] H.C. Lin and CS Raghavendra. A dynamic load-balancing policy with a central job dispatcher (LBC). *Software Engineering, IEEE Transactions on*, 18(2):148–158, 1992. [cited at p. 8, 18, 25]

[56] Michael Mitzenmacher. The power of two choices in randomized load balancing. *Parallel and Distributed Systems, IEEE Transactions on*, 12(10):1094–1104, 2001. [cited at p. 8, 25]

[57] Joanna Kolodziej and Samee Ullah Khan. Multi-level hierarchic genetic-based scheduling of independent jobs in dynamic heterogeneous grid environment. *Information Sciences*, 214:1–19, 2012. [cited at p. 9, 42, 69, 85]

[58] L.Y. Tseng, Y.H. Chin, and S.C. Wang. A minimized makespan scheduler with multiple factors for grid computing systems. *Expert Systems with Applications*, 36(8):11118–11130, 2009. [cited at p. 9, 31]

[59] M.P. Bekakos, G. Gravvanis, E.S. Bazoukis, and K.M. Giannoutakis. Towards a pilot grid platform for internet high performance computations. *Billerica, MA: WIT Press, 2006.*, pages 363–387, 2006. [cited at p. 9]

[60] Y. Li, Y. Yang, M. Ma, and L. Zhou. A hybrid load balancing strategy of sequential tasks for grid computing environments. *Future Generation Computer Systems*, 25(8):819–828, 2009. [cited at p. 9, 69, 70, 72]

[61] M. Choi, J.L. Yu, H.J. Kim, and S.R. Maeng. Improving performance of a dynamic load balancing system by using number of effective tasks. In *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, pages 436–441. IEEE, 2003. [cited at p. 9]

[62] Y. Wang and R. Hyatt. An improved algorithm of two choices in randomized dynamic load-balancing. In *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*, pages 440–445. IEEE, 2002. [cited at p. 9]

[63] J.F. Pineau, Y. Robert, and F. Vivien. The impact of heterogeneity on master-slave scheduling. *Parallel Computing*, 34(3):158–176, 2008. [cited at p. 9, 24]

[64] I. Ahmad, A. Ghafoor, and K. Mehrotra. Performance prediction of distributed load balancing on multicomputer systems. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 830–839. ACM, 1991. [cited at p. 9]

[65] H.D. Karatza. A comparative analysis of scheduling policies in a distributed system using simulation. *International Journal of SIMULATION Systems, Science & Technology*, pages 1–2, 2000. [cited at p. 9]

[66] M. Kafil and I. Ahmad. Optimal task assignment in heterogeneous distributed computing systems. *Concurrency, IEEE*, 6(3):42–50, 1998. [cited at p. 9]

[67] A. Kamthe and S.Y. Lee. A stochastic approach to estimating earliest start times of nodes for scheduling DAGs on heterogeneous distributed computing systems. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005. [cited at p. 9]

[68] GQ Liu, KL Poh, and M. Xie. Iterative list scheduling for heterogeneous computing. *Journal of Parallel and Distributed Computing*, 65(5):654–665, 2005. [cited at p. 9]

[69] M. Jayasinghe, Z. Tari, P. Zeephongsekul, and A.Y. Zomaya. Task assignment in multiple server farms using preemptive migration and flow control. *Journal of Parallel and Distributed Computing*, 71(12):1608–1621, 2011. [cited at p. 9]

[70] Steven Solomon, Parimala Thulasiraman, and Ruppa K Thulasiram. Collaborative multi-swarm pso for task matching using graphics processing units. In *GECCO*, volume 11, pages 1563–1570, 2011. [cited at p. 9]

[71] YS Dai, M. Xie, KL Poh, and GQ Liu. A study of service reliability and availability for distributed systems. *Reliability Engineering & System Safety*, 79(1):103–112, 2003. [cited at p. 10]

[72] George Terzopoulos and Helen Karatza. Power-aware load balancing in heterogeneous clusters. In *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2013 International Symposium on*, pages 148–154. IEEE, 2013. [cited at p. 10]

[73] Hong-bin Wang, Zhi-yi Fang, Guan-nan Qu, and Xiao-dan Ren. An innovate dynamic load balancing algorithm based on task classification. *IJACT: International Journal of Advancements in Computing Technology*, 4(6):244–254, 2012. [cited at p. 10]

[74] Manitpal S. Sidhu, Parimala Thulasiraman, and Ruppa K Thulasiram. A load-rebalance PSO heuristic for task matching in heterogeneous computing systems. In *Swarm Intelligence (SIS), 2013 IEEE Symposium on*, pages 180–187. IEEE, 2013. [cited at p. 10]

[75] K. Lu, R. Subrata, and A.Y. Zomaya. On the performance-driven load distribution for heterogeneous computational grids. *Journal of Computer and System Sciences*, 73(8):1191–1206, 2007. [cited at p. 10]

[76] D. Grosu and A.T. Chronopoulos. Algorithmic mechanism design for load balancing in distributed systems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(1):77–84, 2004. [cited at p. 10, 18, 110, 111]

[77] D. Gu, L. Yang, and L.R. Welch. A predictive, decentralized load balancing approach. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 131b–131b. IEEE, 2005. [cited at p. 10]

[78] K. Li. Optimal load distribution in nondedicated heterogeneous cluster and grid computing environments. *Journal of Systems Architecture*, 54(1):111–123, 2008. [cited at p. 10]

[79] Anastasios A Economides and John A Silvester. A game theory approach to cooperative and non-cooperative routing problems. In *Telecommunications Symposium, 1990. ITS'90 Symposium Record., SBT/IEEE International*, pages 597–601. IEEE, 1990. [cited at p. 10]

[80] X. Qin and T. Xie. An availability-aware task scheduling strategy for heterogeneous systems. *Computers, IEEE Transactions on*, 57(2):188–199, 2008. [cited at p. 10]

[81] Geetika Tewari Lakshmanan, Yuri G Rabinovich, and Robert Jeffrey Schloss. Decentralized load distribution to reduce power and/or cooling costs in an event-driven system, May 11 2013. US Patent App. 13/892,264. [cited at p. 10]

[82] Sandip Chakraborty, Soumyadip Majumder, and Diganta Goswami. Approximate congestion games for load balancing in distributed environment. *arXiv preprint arXiv:1305.3354*, 2013. [cited at p. 10]

[83] S. Abdallah and V. Lesser. Modeling task allocation using a decision theoretic model. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 719–726. ACM, 2005. [cited at p. 10]

[84] Qinma Kang, Hong He, and Huimin Song. Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm. *Journal of Systems and Software*, 84(6):985–992, 2011. [cited at p. 10, 30]

[85] Rajeev Motwani. Lecture notes on approximation algorithms: Volume i. *Dept. Comput. Sci., Stanford Univ., Stanford, CA, Tech. Rep. CS-TR-92-1435*, 1992. [cited at p. 11]

[86] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson Education Inc., 2006. [cited at p. 11, 23, 25, 31, 99, 100, 101, 102]

[87] Dorit S Hochbaum. *Approximation Algorithms for NP-Hard Problems*. Thomson Asia Pte Ltd., 2003. [cited at p. 11, 25, 99, 100, 101]

[88] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra Hensgen, and Sahra Ali. Representing task and machine heterogeneities for heterogeneous computing systems. *Tamkang Journal of Science and Engineering*, 3(3):195–208, 2000. [cited at p. 15]

[89] Alexey Lastovetsky. Special issue of journal of parallel and distributed computing: Heterogeneity in parallel and distributed computing. *Journal of Parallel and Distributed Computing*, 2012. [cited at p. 16]

[90] B Krishna Kumar, S Pavai Madheswari, and KS Venkatakrishnan. Transient solution of an M/M/2 queue with heterogeneous servers subject to catastrophes. *International journal of information and management sciences*, 18(1):63, 2007. [cited at p. 18]

[91] Kishor S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Prentice Hall of India, 2001. [cited at p. 18, 36, 72]

[92] Mitchell D Theys, Tracy D Braun, H. J. Siegal, ANTHONY A Maciejewski, and YK Kwok. Mapping tasks onto distributed heterogeneous computing systems using a genetic algorithm approach. *Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences*, pages 135–178, 2001. [cited at p. 22, 31, 42, 45, 69, 71, 73, 75]

[93] H.D. Karatza and R.C. Hilzer. Load sharing in heterogeneous distributed systems. In *Simulation Conference, 2002. Proceedings of the Winter*, volume 1, pages 489–496. IEEE, 2002. [cited at p. 23, 76]

[94] Shoukat Ali, Tracy D Braun, Howard Jay Siegel, and Anthony A Maciejewski. Heterogeneous computing, 2002. [cited at p. 23]

[95] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001. [cited at p. 23, 24, 31, 36, 42, 45, 69, 76]

[96] XiaoShan He, XianHe Sun, and Gregor Von Laszewski. QoS guided min-min heuristic for grid task scheduling. *Journal of Computer Science and Technology*, 18(4):442–451, 2003. [cited at p. 24, 30, 31]

[97] Hesam Izakian, Ajith Abraham, and Václav Snasel. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, volume 1, pages 8–12. IEEE, 2009. [cited at p. 24, 30, 31]

[98] Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for grid scheduling problems. *Future generation computer systems*, 26(4):608–621, 2010. [cited at p. 24, 31]

[99] G. Attiya and Y. Hamam. Two phase algorithm for load balancing in heterogeneous distributed systems. In *Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on*, pages 434–439. IEEE, 2004. [cited at p. 25, 101]

[100] S. K. Basu. *Design Methods and Analysis of Algorithms*. Prentice Hall of India, 2005. [cited at p. 25, 100]

[101] Scott Kirkpatrick, D. Gelatt Jr., and Mario P Vecchi. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983. [cited at p. 27, 69, 71]

[102] Mohand Mezmaz, Nouredine Melab, Yacine Kessaci, Young Choon Lee, E-G Talbi, Albert Y Zomaya, and Daniel Tuyttens. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing*, 71(11):1497–1508, 2011. [cited at p. 27]

[103] David E. Goldberg and John H. Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988. [cited at p. 27, 68, 83, 84, 86, 87]

[104] Deb Kalyanmoy. *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd., 2004. [cited at p. 27, 76, 87]

[105] David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011. [cited at p. 27]

[106] A.J. Page and T.J. Naughton. Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 189a–189a. IEEE, 2005. [cited at p. 30, 68, 70, 72]

[107] Stefka Fidanova. Simulated annealing for grid scheduling problem. In *Modern Computing, 2006. JVA'06. IEEE John Vincent Atanasoff 2006 International Symposium on*, pages 41–45. IEEE, 2006. [cited at p. 30, 68, 69, 71, 76]

[108] Chuliang Weng and Xinda Lu. Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid. *Future Generation Computer Systems*, 21(2):271–280, 2005. [cited at p. 30]

[109] Joanna Kolodziej, Samee Ullah Khan, Lizhe Wang, and Albert Y Zomaya. Energy efficient genetic-based schedulers in computational grids. *Concurrency and Computation: Practice and Experience*, 2012. [cited at p. 30]

[110] Ellis Horowitz, Sartaj Sahni, and Sanguthevar Rajasekaran. Fundamentals of computer algorithms, 2003. [cited at p. 30, 100, 101]

[111] T.D. Braun, H.J. Siegel, A.A. Maciejewski, and Y. Hong. Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions. *Journal of Parallel and Distributed Computing*, 68(11):1504–1516, 2008. [cited at p. 31, 70]

[112] S.P. Dandamudi. The effect of scheduling discipline on dynamic load sharing in heterogeneous distributed systems. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1997. MASCOTS'97., Proceedings Fifth International Symposium on*, pages 17–24. IEEE, 1997. [cited at p. 31, 42]

[113] S.S. Chauhan and RC Joshi. A weighted mean time min-min max-min selective scheduling strategy for independent tasks on grid. In *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, pages 4–9. IEEE, 2010. [cited at p. 31]

[114] K. Etminani and M. Naghibzadeh. A min-min max-min selective algorihtm for grid task scheduling. In *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*, pages 1–7. IEEE, 2007. [cited at p. 31]

[115] Liya Fan, Fa Zhang, Gongming Wang, Bo Yuan, and Zhiyong Liu. A self-adaptive greedy scheduling scheme for a multi-objective optimization on identical parallel machines. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 43–55. Springer, 2009. [cited at p. 31]

[116] Ming Wu and Xian-He Sun. Grid harvest service: a performance system of grid computing. *Journal of Parallel and Distributed Computing*, 66(10):1322–1337, 2006. [cited at p. 31]

[117] Min-You Wu, Wei Shu, and Hong Zhang. Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pages 375–385. IEEE, 2000. [cited at p. 31]

[118] Manitpal S Sidhu. *A PSO based Load-Rebalance Algorithm for Task-Matching in Large Scale Heterogeneous Computing Systems*. PhD thesis, The University of Manitoba, 2013. [cited at p. 31]

[119] Janos Sztrik. Modelling of heterogeneous multiprocessor systems with randomly changing parameters. *Acta Cybernetica*, 10(1-2):71–84, 1991. [cited at p. 36]

[120] V. Rykov and D. Efrosinin. Numerical analysis of optimal control policies for queueing systems with heterogeneous servers. In *Kalashnikov Memorial Seminar*, 2002. [cited at p. 36]

[121] Tsung-Yin Wang, Jau-Chuan Ke, Kuo-Hsiung Wang, and Siu-Chuen Ho. Maximum likelihood estimates and confidence intervals of an m/m/r queue with heterogeneous servers. *Mathematical Methods of Operations Research*, 63(2):371–384, 2006. [cited at p. 36]

[122] Ilias Iliadis and LY-C Lien. Resequencing delay for a queueing system with two heterogeneous servers under a threshold-type scheduling. *Communications, IEEE Transactions on*, 36(6):692–702, 1988. [cited at p. 36]

[123] V. Rykov and D. Efrosinin. On performance characteristics for queuing systems with heterogeneous servers. volume 69, pages 61–75, 2008. [cited at p. 36]

[124] Y.C. Chow and W.H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *Computers, IEEE Transactions on*, 100(5):354–361, 1979. [cited at p. 36]

[125] Rajeev Motwani. *Randomized algorithms*. Cambridge university press, 1995. [cited at p. 43]

[126] W David Kelton and Averill M Law. *Simulation modeling and analysis*. McGraw Hill Boston, MA, 2000. [cited at p. 44]

[127] Natallia Kokash. An introduction to heuristic algorithms. *Department of Informatics and Telecommunications*, 2005. [cited at p. 68]

[128] Zbigniew Michalewicz. *Genetic algorithms+ data structures= evolution programs*. springer, 1996. [cited at p. 68, 80, 83, 86, 87]

[129] Kenneth A. De Jong and William M. Spears. Using genetic algorithms to solve NP-complete problems. In *ICGA*, pages 124–132, 1989. [cited at p. 68]

[130] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994. [cited at p. 69, 86, 87, 88]

[131] W.A. Greene. Dynamic load-balancing via a genetic algorithm. In *Tools with Artificial Intelligence, Proceedings of the 13th International Conference on*, pages 121–128. IEEE, 2001. [cited at p. 69, 70]

[132] A.J. Page, T.M. Keane, and T.J. Naughton. Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system. *Journal of parallel and distributed computing*, 70(7):758–766, 2010. [cited at p. 69, 70]

[133] J. Aguilar and E. Gelenbe. Task assignment and transaction clustering heuristics for distributed systems. *Information Sciences*, 97(1):199–219, 1997. [cited at p. 69, 70]

[134] Deo Prakash Vidyarthi and Anil Kumar Tripathi. Maximizing reliability of distributed computing system with task allocation using simple genetic algorithm. *Journal of Systems Architecture*, 47(6):549–554, 2001. [cited at p. 69, 70]

[135] Kalyanmoy Deb. An introduction to genetic algorithms. *Sadhana*, 24(4-5):293–315, 1999. [cited at p. 69, 87]

[136] Ajith Abraham, Rajkumar Buyya, and Baikunth Nath. NatureŠs heuristics for scheduling jobs on computational grids. In *The 8th IEEE international conference*

on advanced computing and communications (ADCOM 2000), pages 45–52, 2000. [cited at p. 69]

[137] Shan Jin and Wei Zhou. LBSG: A load balancing scenario based on genetic algorithm. In *Emerging Technologies for Information Systems, Computing, and Management*, pages 349–356. Springer, 2013. [cited at p. 69]

[138] Jung-Ug Kim and Yeong-Dae Kim. Simulated annealing and genetic algorithms for scheduling products with multi-level product structure. *Computers & Operations Research*, 23(9):857–868, 1996. [cited at p. 69]

[139] E-G Talbi and Traian Muntean. Hill-climbing, simulated annealing and genetic algorithms: a comparative study and application to the mapping problem. In *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, volume 2, pages 565–573. IEEE, 1993. [cited at p. 69, 71]

[140] Amir Masoud Rahmani and Mojtaba Rezvani. A novel genetic algorithm for static task scheduling in distributed systems. *International Journal of Computer Theory and Engineering*, 1(1):1793–8201, 2009. [cited at p. 69]

[141] A.J. Page and T.J. Naughton. Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. *Artificial Intelligence Review*, 24(3):415–429, 2005. [cited at p. 70, 89]

[142] Mona Aggarwal, Robert D Kent, and Alioune Ngom. Genetic algorithm based scheduler for computational grids. In *High Performance Computing Systems and Applications, 2005. HPCS 2005. 19th International Symposium on*, pages 209–215. IEEE, 2005. [cited at p. 70]

[143] J. Carretero, F. Xhafa, and A. Abraham. Genetic algorithm based schedulers for grid computing systems. *International Journal of Innovative Computing, Information and Control*, 3(6):1–19, 2007. [cited at p. 70]

[144] I. Ahmad, M.K. Dhodhi, and A. Ghafoor. Task assignment in distributed computing systems. In *Computers and Communications, 1995. Conference Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on*, pages 49–53. IEEE, 1995. [cited at p. 70]

[145] K.M. Yu and C.K. Chen. An evolution-based dynamic scheduling algorithm in grid computing environment. In *Intelligent Systems Design and Applications, 2008.*

ISDA'08. Eighth International Conference on, volume 1, pages 450–455. IEEE, 2008. [cited at p. 70]

[146] Mohammad Hassan Shenassa and Mahdi Mahmoodi. A novel intelligent method for task scheduling in multiprocessor systems using genetic algorithm. *Journal of the Franklin Institute*, 343(4):361–371, 2006. [cited at p. 70]

[147] S.H. Lee and C.S. Hwang. A dynamic load balancing approach using genetic algorithm in distributed systems. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 639–644. IEEE, 1998. [cited at p. 70]

[148] C.W. Cheong and V. Ramachandran. Genetic based web cluster dynamic load balancing in fuzzy environment. In *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, volume 2, pages 714–719. IEEE, 2000. [cited at p. 70]

[149] B. Ucar, C. Aykanat, K. Kaya, and M. Ikinci. Task assignment in heterogeneous computing systems. *Journal of parallel and Distributed Computing*, 66(1):32–46, 2006. [cited at p. 71, 75]

[150] Balram Suman and Prabhat Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the operational research society*, 57(10):1143–1160, 2005. [cited at p. 71, 76]

[151] K Krishna, K Ganeshan, and D Janaki Ram. Distributed simulated annealing algorithms for job shop scheduling. *Systems, Man and Cybernetics, IEEE Transactions on*, 25(7):1102–1109, 1995. [cited at p. 71]

[152] Shaoqiang Zhang, Huazhi Sun, Guojun Li, and Zhengchang Su. A simulated annealing heuristic for resource allocation and sceduling with precedence constraints. *Information Computational Science*, 8(9):1541–1550, 2011. [cited at p. 71]

[153] Arnold O. Allen. *Probability, Statistics, and Queuing Theory with Computer Science Applications*. Academic Press, second edition, 2005. [cited at p. 72]

[154] The MathWorks, Inc.Ţ, Natick, M.A. *Genetic algorithm and Direct Search Toolbox: Users Guide*, version 2.4.1 edition, 2009. [cited at p. 80, 83]

[155] David E. Goldberg. *Genetic Algorithms in search, Optimization and Machine Learning*. Pearson Education, 2002. [cited at p. 83]

[156] M. Munetomo, Y. Takai, and Y. Sato. A genetic approach to dynamic load balancing in a distributed computing system. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 418–421. IEEE, 1994. [cited at p. 83]

[157] Randy L. Haupt and Sue Ellen Haupt. *Practical Genetic Algorithm*. John Wiley and Sons Inc., second edition, 2004. [cited at p. 87]

[158] Anany Levitin. *Introduction To Design And Analysis Of Algorithms, 2/E*. Pearson Education India, 2008. [cited at p. 100, 104]

[159] Carla P Gomes and Ryan Williams. Approximation algorithms. In *Search Methodologies*, pages 557–585. Springer, 2005. [cited at p. 100]

[160] Narendhar Maaroju, KVR Kumar, and Deepak Garg. Approximation algorithms for NP-problems. *CIIT International Journal of Software Engineering and Technology, Issue May-2009 ISSN*, 1(2):7–11, 2009. [cited at p. 100, 102]

[161] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1):259–271, 1990. [cited at p. 101]

[162] Reuven Cohen, Liran Katzir, and Danny Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, 100(4):162–166, 2006. [cited at p. 101, 102]

[163] P. Schuurman and G.J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999. [cited at p. 101]

[164] D.B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1):461–474, 1993. [cited at p. 101]

[165] Nikhil R Devanur, Kamal Jain, Balasubramanian Sivan, and Christopher A Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. In *Proceedings of the 12th ACM conference on Electronic commerce*, pages 29–38. ACM, 2011. [cited at p. 102]

[166] N. Alon, Y. Azar, G.J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998. [cited at p. 102]

[167] Li-Chuan Chen and Hyeong-Ah Choi. Approximation algorithms for data distribution with load balancing of web servers. In *Proc. International Conference on Cluster Computing (CLUSTER 2001)*, 2001. [cited at p. 102]

[168] Gruia Calinescu, Amit Chakrabarti, Howard Karloff, and Yuval Rabani. Improved approximation algorithms for resource allocation. *Integer Programming and Combinatorial Optimization*, pages 401–414, 2006. [cited at p. 102]

[169] Ravi Varadarajan. An efficient approximation algorithm for load balancing with resource migration in distributed systems. *manuscript. University of Florida*, 1992. [cited at p. 102]

[170] Fabian A Chudak and David B Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 581–590. Society for Industrial and Applied Mathematics, 1997. [cited at p. 102]

[171] Luiz F Bittencourt, Flávio Keidi Miyazawa, and André L Vignatti. Distributed load balancing algorithms for heterogeneous players in asynchronous networks. *J. UCS*, 18(20):2771–2797, 2012. [cited at p. 110, 118]

[172] R. Subrata, A.Y. Zomaya, and B. Landfeldt. Game-theoretic approach for load balancing in computational grids. *Parallel and Distributed Systems, IEEE Transactions on*, 19(1):66–76, 2008. [cited at p. 110, 113]

[173] Karl Tuyls and Simon Parsons. What evolutionary game theory tells us about multiagent learning. *Artificial Intelligence*, 171(7):406–416, 2007. [cited at p. 110]

[174] Noam Nisan. *Algorithmic game theory*. Cambridge University Press, 2007. [cited at p. 110]

[175] Satish Penmatsa and Anthony T Chronopoulos. Cooperative load balancing for a network of heterogeneous computers. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006. [cited at p. 110]

[176] John Sutton. Non-cooperative bargaining theory: An introduction. *The Review of Economic Studies*, 53(5):709–724, 1986. [cited at p. 110]

[177] D. Grosu and A.T. Chronopoulos. A truthful mechanism for fair load balancing in distributed systems. In *Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on*, pages 289–296. IEEE, 2003. [cited at p. 111]

[178] D. Grosu, A.T. Chronopoulos, and M.Y. Leung. Load balancing in distributed systems: An approach using cooperative games. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 52–61. IEEE, 2002. [cited at p. 111]

[179] Eyal Even-Dar, Alex Kesselman, and Yishay Mansour. Convergence time to nash equilibrium in load balancing. *ACM Transactions on Algorithms (TALG)*, 3(3):32, 2007. [cited at p. 111]

[180] Preetam Ghosh, Nirmalya Roy, Sajal K Das, and Kalyan Basu. A pricing strategy for job allocation in mobile grids using a non-cooperative bargaining theory framework. *Journal of Parallel and Distributed Computing*, 65(11):1366–1383, 2005. [cited at p. 112]

[181] Benjamin Yolken and Nicholas Bambos. Game based capacity allocation for utility computing environments. *Telecommunication Systems*, 47(1-2):165–181, 2011. [cited at p. 112]

[182] John Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951. [cited at p. 112]

[183] W Davis Dechert. Non cooperative dynamic games: a control theoretic approach. *University of Houston, URL: http://algol. ssc. wisc. edu/research/research/dgames. pdf*, 1997. [cited at p. 112]

[184] D. Grosu and A.T. Chronopoulos. Noncooperative load balancing in distributed systems. *Journal of Parallel and Distributed Computing*, 65(9):1022–1034, 2005. [cited at p. 112, 115, 117, 118, 124, 129]

[185] S. Penmatsa and A.T. Chronopoulos. Dynamic multi-user load balancing in distributed systems. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10. IEEE, 2007. [cited at p. 112]

[186] E. Altman, U. Ayesta, and B.J. Prabhu. Optimal load balancing in processor sharing systems. In *In Proc. of GameComm*. Citeseer, 2008. [cited at p. 112]

[187] Paul Frihauf, Miroslav Krstic, and Tamer Basar. Nash equilibrium seeking in noncooperative games. *Automatic Control, IEEE Transactions on*, 57(5):1192–1207, 2012. [cited at p. 112]

[188] Samee Ullah Khan and Ishfaq Ahmad. Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006. [cited at p. 112]

[189] Wang Yunzhi, Hu Yaoguang, Jia Yanhua, and Zhang Ruijun. A game theoretic approach to product-mix resource allocation. In *Industrial Electronics and Applications (ICIEA), 2010 the 5th IEEE Conference on*, pages 2142–2147. IEEE, 2010. [cited at p. 117]

[190] Satish Penmatsa and Anthony T Chronopoulos. Job allocation schemes in computational grids based on cost optimization. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005. [cited at p. 119, 120, 129]

[191] Ajay D Kshemkalyani and Mukesh Singhal. *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2008. [cited at p. 120]

[192] P. Berenbrink, T. Friedetzky, L.A. Goldberg, P. Goldberg, Z. Hu, and R. Martin. Distributed selfish load balancing. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 354–363. ACM, 2006. [cited at p. 124]

Thesis contribution

Conferences

1. Bibhudatta Sahoo, S. Mahapatra, and S.K. Jena, "Genetic Algorithm Based Dynamic Load Balancing Scheme for Heterogeneous Distributed Systems ", *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, PDPTA 2008, Las Vegas, Nevada, USA, July 14-17, 2008, 2 Volumes. CSREA Press 2008, ISBN 1-60132-084-1.

2. Bibhudatta Sahoo, P.Kumar Chandra, and S.K. Jena, "Modeling and Analysis to Estimate the Performance of Heterogeneous Cluster ", *50th Technical Review of Institution of Engineers*, pp. 113-118, February 2009.

3. P.Kumar Chandra,and Bibhudatta Sahoo, "Dynamic load distribution algorithm performance in heterogeneous distributed system for I/O- intensive task ", *TENCON 2008 IEEE Region 10 Conference*, pp. 1-5, 19-21 Nov. 2008.

4. Bibhudatta Sahoo, S.K. Jena and S. Mahapatra, "Load Balancing in Heterogeneous Distributed Computing Systems using Approximation Algorithm ", *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, PDPTA 2013, Las Vegas, Nevada, USA, July 22-25, 2013, 2 Volumes. CSREA Press 2013, ISBN 1-60132-084-1.

Book Chapter

1. Bibhudatta Sahoo, Sanjay Kumar Jena, and Sudipta Mahapatra, "Heuristic Resource Allocation Algorithms for Dynamic Load Balancing in Heterogeneous Distributed Computing System", in *Advances in Secure Computing, Internet Services, and Applications* ,IGI Global, 2014, Web. 3 Nov. 2013. doi:10.4018/978-1-4666-4940-8.

Journal

1. Bibhudatta Sahoo, Sanjay Kumar Jena, and Sudipta Mahapatra, "Simulated Annealing based Heuristic Approach for Dynamic Load Balancing Problem on Heterogeneous Distributed Computing System", *International Journal of Artificial Intelligent System and Machine Learning* Vol.7 No.2, pp. 65 - 75, March, 2013.

2. Bibhudatta Sahoo, Dilip Kumar, and Sanjay Kumar Jena, "Analysing the Impact of Heterogeneity with Greedy Resource Allocation Algorithms for Dynamic Load Balancing in Heterogeneous Distributed Computing System ",*International Journal of Computer Applications* Vol.62 No.19, pp. 25 - 34, January 2013.

3. Bibhudatta Sahoo, Sanjay Kumar Jena, and Sudipta Mahapatra, "Game Theory based Improved Job Allocation Schemes for Heterogeneous Distributed Computing System with Optimal Load Fractions", *Communicated to International Journal Computer Sc. and Information System by ComSIS Consortium.*

4. Bibhudatta Sahoo, Sanjay Kumar Jena, and Sudipta Mahapatra, "A simulation framework for centralized greedy load balancing algorithms in Heterogeneous Distributed Computing System ", *Communicated to The journal Simulation Modelling Practice and Theory* , Elsevier, http://www.journals.elsevier.com/simulation-modelling-practice-and-theory.